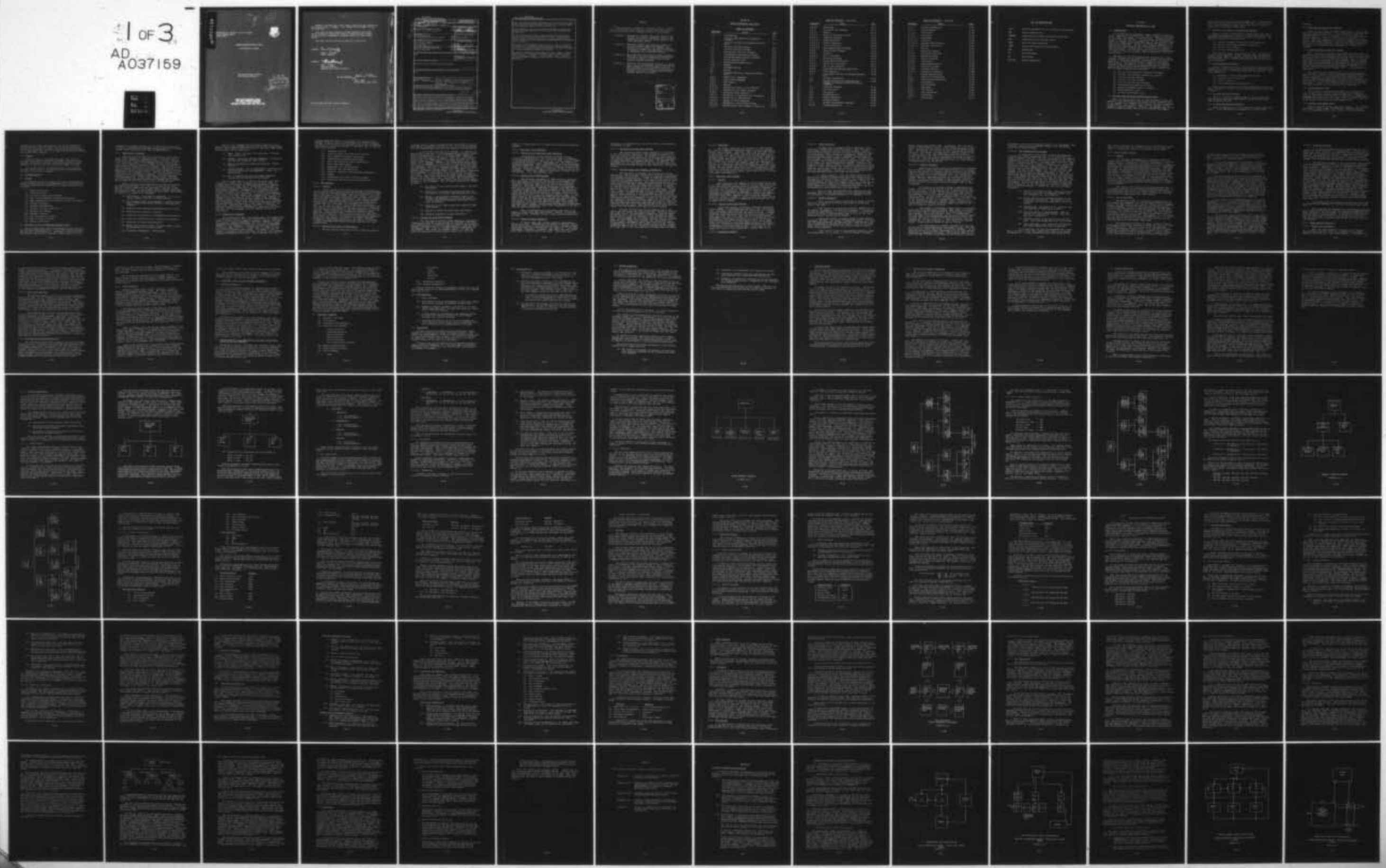
AD-A037 159 NORTH ELECTRIC CO GALION OHIO F/G 17/2
COMMUNICATIONS PROCESSOR SYSTEM.(U)
JAN 77 K HAGSTROM, B BEIZER F30602-73-C-0314
UNCLASSIFIED RADC-TR-76-394 NL

1 OF 3
AD
A037159



AD A037159

(12)

RADC-TR-76-394, Volumes VI & VII (of eight)
Final Technical Report
January 1977

COMMUNICATIONS PROCESSOR SYSTEM

North Electric Company

Approved for public release;
distribution unlimited.

DDC
RECEIVED
MAR 18 1977
A

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

Because of the small size of each volume, volumes have been combined into the following reports: Volumes I - III, Volumes IV & V, Volumes VI & VII, and Volume VIII.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and approved for publication.

APPROVED: [illegible]

DANIEL J. McAULIFFE
Project Engineer

APPROVED: [illegible]

FRED I. DIAMOND
Technical Director
Communications & Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER 18 RADC-TR-76-394, ~~Vols VI & VII~~ (of eight) VI-VII	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 *COMMUNICATIONS PROCESSOR SYSTEM.		5. TYPE OF REPORT & PERIOD COVERED 9 Final Technical Report. June 1973 - March 1976, 6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) 10 Mr. Kenneth Hagstrom Dr. Boris Beizer, Data Systems Analysts		8. CONTRACT OR GRANT NUMBER(s) 15 F30602-73-C-0314
9. PERFORMING ORGANIZATION NAME AND ADDRESS North Electric Company 553 South Market Street Galion OH 44833		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 16 45191904 17 19
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (DCLT) Griffiss AFB NY 13441		12. REPORT DATE 11 January 1977 13. NUMBER OF PAGES 265 12 276p.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

V8 A036 874

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Daniel J. McAuliffe (DCLT)

Because of the small size of each volume, volumes have been combined into the following reports: Volumes I - III, Volumes IV & V, Volumes VI & VII, and Volume VIII.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Communications Switching, Communications Processors, Processor Architecture, Circuit Switching, Message Switching, Packet Switching, Base Distribution

259 750

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report covers the results of a study to develop a hardware architecture which will be the basis for a family of communications processors for applications processors for application in circuit, message, packet and base communications switch configurations. Over 23 switching equipments were investigated from which a functional baseline was defined for use in the subsequent studies for evolving an advanced Communications Processor System (CPS) architecture. These switches included circuit, message, and packet switching equipments which were felt to be typical of traditional and advanced switching concepts. As an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

bpg

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

example, the AN/TTC-39, AUTODIN, ARPA Network were used as part of the circuit, message and packet switching baselines respectively. Air Force Base Communications studies were used as the baseline in that area.

From this baseline, a set of fifteen primitive functions were derived which represent the needed capabilities for any generally applied communications processor (CP).

The latest in the state-of-the-art in ADP technology was investigated to determine the best and most viable approach to the CPS. The goal being to develop a family of CP's which could be used to satisy switching needs for circuit, message, packet, base communications applications or in an integrated node of the future.

The results of the investigation were continually the subject of trade-offs through the use of an analytical modelling technique. The final outcome of this effort is a ten part specification detailing the performance requirements of each unit comprising the communications processor.

*This report is organized into eight volumes as follows: Volume I - Executive Summary; Volume II - Definition of Problem; Volume III - Modelling; Volume IV - CCC Architecture; Volume V - CPS Architecture; Volume VI - Software Generation; Volume VII - Appendices; and Volume VIII - CPS Central Processor Specification.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

PREFACE

Technical Report, Volume VI is divided into four sections, each section addressing a specific area of the total communications switching system software generation requirements.

Section 1 - Categorizes software components from an operational requirements standpoint, defines the categories, and assesses their impact on the total operating system.

Section 2 - Describes methods for reducing overall cost associated with software development and writing, and identifies the support packages (including software test facilities) necessary to implement the recommended approach.

Section 3 - Describes the system architectural considerations that must be evaluated and implemented in order to effect a consolidated overall systems software approach.

Section 4 - Addresses the requirements necessary to identify, select, and configure Circuit and Message Switch software over a variety of physical and functional environments, and describes the facility necessary to implement the reusable software package philosophy.

Volume VII consists of five appendices.

ACCESSION for

NTIS White Section [✓]

DDC Buff Section ☐

UNANNOUNCED ☐

JUSTIFICATION ....

BY

DISTRIBUTION/AVAILABILITY CODES

Dist. AVAIL. and/or SPECIAL

A

VOLUME VI

SYSTEM GENERATION FACILITIES

TABLE OF CONTENTS

TABLE OF CONTENTS - Continued

TABLE OF CONTENTS - Continued

LIST OF ABBREVIATIONS

ASCII	American Standard Code for Information Interchange
BCD	Binary Coded Decimal
COMSDOS	Comprehensive Software Development Operating
CPS	Communications Processing System
DTMF	Dual Tone Multi-Frequency
GASM	Generalized Analytical System Model
HEX	Hexidecimal
MF	Multi-Frequency
ms	millisecond
SYS GEN	Systems Generation

SECTION 1

SOFTWARE COMPONENTS OF A CPS

1.0 INTRODUCTION

It is tempting to recommend a single, simplistic approach to the reduction of software costs, such as "higher level languages", "modular software", "reentrant code", "decision tables", or some such other nonsense. Any such single minded panacea is doomed to failure since it presumes that all the software components of a communication system are similar and amenable to the same kind of treatment. In fact, the software components of a Communication Processing System (CPS) represent a diversity as wide as the entire software field. One would not recommend "decision tables", say for equal application to the construction of a matrix inversion program, a payroll program, and an operating system.

The point of view taken here is not to search for a non-existent strategy (in the game theoretic sense) but for a mixed strategy of manifold components, each applied in the proper proportion to the particular kind of communication software being written. The purpose of this section is to establish software categories that are distinguished from the point of view of the tools that are best suited to their construction. The following categories are convenient:

(1) On-line, high execution probability software.

(2) On-line, low execution probability software.

(3) Critical on-line software.

(4) Critical off-line software.

(5) Non-critical off-line software.

(6) Configuration generation software.

(7) System performance monitor software.

(8) System analysis tools.

(9) Software development tools.

1.1 On-Line, High Execution Probability Software

What is actually meant here, is that software in which the product of the probability of execution and the resources used by the programs is a significant fraction of the total resources of the system. Such software, must, for the present, be programmed in machine language. Therefore, the emphasis must be on creating common software and the use of software development tools that will alleviate the programming costs. Approximately 40%-50% of the on-line software of a CPS falls into this category. This comprises, perhaps, 25% of the total software associated with a communication system.

1.2 On-Line, Low Execution Probability Software

This is on-line software which, despite large size, has a low execution probability and does not therefore statistically contribute significantly to the utilization of resources. Examples of this kind of software are:

(1) Operator command interpretation and execution.

(2) Traffic and system management functions.

(3) On-line table changes.

(4) On-line report generation.

Such software could, with little sacrifice, be written in a higher level language. In many systems, these programs are surged, meaning that some degree of inefficiency has already been accepted. This category comprises some 30% to 40% of the on-line programs.

1.3 Critical On-Line Software

This is software which may or may not have a high execution probability or may or may not be statistically significant, but which require a lot of timeliness when they are executed. Typical examples are:

(1) Emergency resources management (overflow).

(2) System recovery functions.

(3) Traffic recovery.

(4) Other emergency processing modes.

These must be treated as the high probability on-line software. Perhaps 5% to 10% of the on-line software falls into this category.

1.4 Critical Off-Line Software

This is a relatively small category. Off-line software, with critical timeliness requirements, should probably have been mechanized on-line.

1.5 Non-Critical Off-Line Software

This is a grab bag of off-line software not discussed elsewhere. It can and should be programmed in a higher level language.

1.6 Configuration Generation Software

There are two primary elements to this group: the configurator and the table generator. The configurator establishes the program parameter values for the particular site, given the characteristics of the site hardware. It also performs routine resource analysis to see to it that the site has what it requires. The table generator is a form of a higher level language processor which takes the source table information and converts it to the object tables. Both of these could and should be mostly site independent and programmed in a higher level language.

1.7 System Performance Monitor Software

Here, a distinction must be made between on-line performance monitoring and performance monitoring under test conditions. Sophisticated on-line performance monitoring cannot be done since it tends to cut into the systems performance. That is, significant artifact is introduced if the performance monitor consumes more than a small percentage of the resources. On-line performance monitors would tend to be thoroughly integrated into the on-line high execution probability software. Therefore, it would be written in machine language and it is programming tools that matter.

Test condition performance monitor tools can be allowed more artifact. However, since the programs being measured are close to resources (in the sense that they manipulate and allocate resources) it is not likely that the uncertainty and excessive processing introduced by a higher level language would be tolerated. If the performance monitor tool is of a general type (e.g., a trace), then, since it will be written only once, the means by which it is constructed is not important.

1.8 System Analysis Tools

This category of programs need never be executed at the site, either on or off-line. This software can be made completely independent and need never be rewritten. A higher level language will suffice for such tools. Part of the problem has been that such tools have not always been available, or have had to be written from scratch for each application.

1.9 Software Development Tools

This is probably the most important category. All language processors, traces, dumps, flowcharters, library utilities, etc., are included in this area. Again, it is not how these programs are developed that matters, but the availability of such programs in order to reduce the cost of developing the operational and support software. This package is again application independent and can be (but would not usually be) written in a higher level language.

1.10 Summary

The first part of the grand strategy, then, that of writing less software is directly applicable to the off-line software, the support software, and the non-critical on-line software, as well as the list of potential common routines.

The second part of the grand strategy, that of making software easier to write, is applicable to the remainder and the bulk of the on-line software.

2.0 SOFTWARE WRITING

2.1 General

Considering the entire software activity, from analysis to coding, through testing, and maintenance, it is found that the following strategies will contribute to making software easier to write:

(1) Reduce source bugs.

(2) Find bugs faster.

(3) Localize the bugs when detected.

(4) Make inter-programmer communications easier.

(5) Make standards easier to define, follow, and enforce.

(6) Automatic analysis.

(7) Automate documentation.

(8) Automate test data generation.

(9) Automate test design.

(10) Automate testing.

(11) Make bugs easier to catch.

(12) Make it easier to patch.

(13) Reduce source bugs.

2.2 Available Tactics In Reducing Software Costs

The following tactics have been employed in the past to reduce software development costs. None of the items are wholly new, but their interworking is. Furthermore, while these have been around, they have not, generally, been available to the designers of switching systems, who for the most part have had to work in a programming environment that has not, until recently, changed significantly from the days of CONUS AUTODIN.

2.2.1 Higher Level Languages

The usual way one starts off a discussion of the role of higher level languages in communications, (or for that matter, in any other specialized application area), is with a long diatribe on why the existing 358 languages are not suited to the task. Having done this, it is followed up with a specification of desirable technical features, which is in turn followed by a formal syntactic definition of the language. What we have at the end, is language 359, which will be partially implemented on an experimental basis, but which will be largely ignored by the very community it was intended to serve. This will be blamed on the recalcitrance of programmers in general, poor presentation of the language at the local computer conference, and most important, a bad implementation. A year later, another group, attacking the same problem will make equally nasty comments about language 359 in order to justify the construction of language 360.

That's the way its been and there is no reason to believe that <u>this</u> specification, (if it were to be done in the above manner), would be singularly more successful. Why has it gone this way in the past? It is, perhaps, that the wrong things have been expected or that the wrong questions were asked. What should be expected of a higher level language?

(1) Universality - its usage is widespread. It is an accepted language with lots of practitioners.

(2) The language lowers the programmer's training requirements, thereby increasing the base of available programmers.

(3) Simplifies the definition and manipulation of files.

(4) Simplifies the creation of reports.

(5) Simplifies the creation of algorithms and processes.

(6) Allows the construction of a library of structures.

(7) Takes care of other routine details.

(8) Checks source syntax errors - provides default values for ambiguities in the source code.

(9) Is machine independent - transportable.

While no one language has all of these features to the desired degree, many languages do have most of these capabilities. However, some languages are more suited to some areas than others. The following list is instructive:

(1) COBOL - Report writing, file structures, universal, tape and disc I/O.

(2) FORTRAN - Universal, machine independent, processing and algorithm building, large library.

(3) MARK-IV (Informatics proprietary program). Report generator.

(4) CODASYL DDL/DML - Not yet implemented, excellent data structure definition and management. Fancy file manipulation.

(5) PL-1 - Probably the best all around compromise, lacking only universality and acceptance.

What it comes down to, is that if the object code inefficiencies of higher level languages are accepted, there is probably more to be gained by using existing popular languages, or modifications of them, than by attempting to promulgate a specialized language whose usage will be even more restricted than assembly language. Furthermore, it is probable that much of what is gained out of a higher level language can be more effectively obtained by other means - e.g., by a good assembler. This is not to say that higher level languages have no place in communications, but that the focus on higher level languages to implement specialized functions may prevent the development of other tools which could be more cost effective. It is, perhaps, more advantageous to have these tools fully exploited first and thereby gain more experience as to what might be desirable in a higher level communication language, and also, to try using existing languages for those functions to which they are suited. Only then, if a new language is still required, go into its construction.

2.2.2 Lower Level Languages

The assembler has been, for the most part, the orphan child of the software package. Since most applications in a commercial computer line are not programmed in assembly language, and since most assembly language programmers are used to having it tough, assemblers have changed but little since the days of AUTOCODER (UNIVAC-I). The assembler is probably the most neglected tool in the programmer's kit bag. In many cases, it lacks even the most primitive diagnostic facilities. The crudity of the assembler is probably a major contributor to the fact that assembly language programming is difficult. The following is a list of features deemed desirable in an assembler for communication language software. Most of the features are not new or unique and most of the features have appeared in one or more assemblers. Unfortunately, the entire package, or something like it, has not always been available.

(1) Assembler controls.

(2) Cross reference lists and their closures.

(3) Data element definitions and labels.

(4) Independently assembled subroutines.

(5) Macro call and macro define capabilities.

(6) Local and global labels.

(7) Subroutine call and definition.

(8) Generator call and definition.

(9) Absolute, relative, and relocatable addressing.

(10) Common.

(11) Integration with other software packages.

(12) Protection and restrictions specifications.

2.2.3 <u>Flow Charter</u>

2.2.3.1 <u>General</u>

A flow charter is a program which uses a combination of source program syntax analysis and cues provided by the programmer, also as part of the source code and uses this information to produce a flow chart of the program. The flow charter is not infallible and cannot represent everything that is coded, but for that matter neither can a human flow charter. Flow charter programs were originally produced primarily as an aid to documentation. In many cases, flow charters have been abused by using them to generate the only flow charts ever produced for the project. That is, the code is used to generate the flow charts rather than vice versa. The true purpose of the flow charter is to maintain the flow charts current with the inevitable modifications of the programs. Initial design flow charts are produced as usual. These are replaced when coding is done, by the flow chart produced by the flowcharter. As program modifications are made, the flow charter is re-run. It eliminates or reduces discrepancies between the code and the flow charts, and equally important, eliminates a discouraging period after the project is all over, in which programmers must spend many hours updating the final documentation.

2.2.3.2 <u>General Facilities of Flow Charter</u>

The flow charter considered for this software system is a package which is wholly integrated with the assembler, and in fact, may be considered to be part of it. It is also integrated with the timing analyzer and the test organizer described below.

A one-for-one flow charter is useless. It requires too much effort on the part of the programmer and gives him an overly detailed representation. In all, he would rather examine the source code than look at that kind of flow chart. Therefore, a many-for-one flow charter is needed. There is a basic question in the design of a flow charter - how smart should it be. In higher level languages, because of the rigid syntax and stratification of the language, it is possible to build reasonably smart flow charters which require very little in the way of cues from the programmer. However, in assembly language, in which indexed, indirect operations, execute instructions, and other forms of address modification can take place, it is impossible to guarantee the development of the proper flow on the basis of the source code alone. It is never really clear where a jump is going to, and in fact, many of the processes as coded cannot be represented directly in a flow chart form. Accordingly, the overwhelming majority of the direction of the flow to the flow charter should be given by the programmer. The following facilities are deemed desirable.

(1) Two levels of flow charting specifiable - detailed and general.

(2) Definition of processes that span more than one instruction, or possibly jump instructions as well.

(3) Option to automatically include all macro, subroutine, and generator calls (one level only), or to delete such calls and imbed them within a process, if desired.

(4) Insertion of all labels which are targets of jump instructions.

(5) Ability to define and flow chart jump tables.

(6) Automatic generation of page connectors.

2.2.3.3 <u>Integration with Other Packages</u>

The flow charter outputs include diagnostic information regarding the topology of the program not available from assembler outputs alone. In addition, the flow charter can be used to produce a representation of the program based on connected nodes and spanning links - that is, a graph theoretic description of the program. This output is extremely useful. It forms the basis for automating a timing analysis model and is also the basis for a model which will be used by the test data generation package.

2.2.4 Structural Test Generator

2.2.4.1 Definition of Structural Test Generator

A distinction is made between two kinds of tests - structural and functional. In a structural test, the emphasis is on the program and its logic, but not on what it does in terms of switching functions. In a functional test, the emphasis is completely functional, and is independent of the way the program has been implemented. While there is a degree of overlap between these two, there are enough differences to warrant different approaches. Structural testing is that which is mostly done by the programmers. Functional testing is that which is mostly done by the end user - e.g., an acceptance test.

2.2.4.2 Definition of 100% Testing

The universal employment of 100% testing is advocated. What this means is that every instruction in the program will have been executed at least once, and that every conditional branch will have been taken at least once in every possible direction. Given the topological description of the program, that is, its flow chart, obtaining a minimum sized 100% test is a straight-forward task. The process begins at the program entrance and finds the shortest path to the exit. Any decisions along the way are noted and data must be prepared that will condition the decision so that the indicated path is taken. It then starts again at the entrance and finds the next shortest path. Some of the data on this path may coincide with data of the previous path, so that only that information required to condition the decisions to the new path must be provided. The process is continued in this manner, tracing paths and supplying conditioning data and output information, until every instruction in the program has been accounted for.

This is substantially the procedure used today in the design of large scale real-time systems. However, in practice, paths are missed, redundant testing is done and good records are not kept.

2.2.4.3 Extent of Effort Required

It can be shown that testing comprises approximately 50% of the effort. Consider now two aspects of testing; setting up the tests (phase 1) and executing the test, finding the bugs, correcting them, and re-running the test (phase 2). It is found that phase 1 comprises, in general, some 75% of the test effort, while phase 2 requires 25% of the effort. The effort then, should be on the generation of the tests and not the execution and removal of bugs.

2.2.4.4 Automation of Structural Testing

The topological model output produced as a by-product of the flow charter contains all the information required to trace and sensitize test paths. The process of extracting the test data and the predicted results can be automated through an interactive program. Setting up test run files and comparison of output data can be automated completely. In addition, the maintenance of the test data files, and the editing thereof require facilities already present in most commercial time-sharing systems.

2.2.4.5 The Structural Test Package in Operation

The completion of assembly for a given unit and the generation of the flow chart is the point at which the structural test package can be invoked. The unit is examined to determine all required precondition data. That is, that data which is contained in the calling sequence or in common, which must be specified for the program to be run. The first sensitized path is chosen and the flow chart comments associated with the decisions on that path are printed out, along with a request to specify the variable names for which values must be given. A check is made to see if all unbounded variables have been bound. Values for all inputs are requested and a file is constructed for the input values. A request is made for names of output variables and the values which are to be associated with the given test. However, for the next test, a new file is created and only those input values which have changed as a result of the new test path are requested. Similarly, another output file is created. As the test generation continues, less and less explicit inputs are required since more and more parameters and values are carried over from previous tests. When the entire set of paths have been traced, the test specification phase has been finished.

The user can then call for an automatic run or can direct the test system to a particular test. During an automatic run, each test will be performed in sequence, using the appropriate data file. Any discrepancy between the supplied output and the actual output will indicate a test failure and cause a hold. The programmer can then either continue with the sequence, rerun the test, obtain other outputs, edit the program, etc. After a round of debugging, using trace facilities or whatever is appropriate, the entire test sequence can be rerun. Simple checks are made to see to it that the topology has not changed. If it has, part of the test generation procedure must be redone.

2.2.4.6 Advantages

The primary advantage is a guarantee of 100% testing. But, there are additional advantages which cannot be obtained without such a package. Typically, today even when 100% testing is used, after a bug is found, the entire test procedure is not rerun from the top. While this can often be done with safety, no small number of bugs remain which could have been caught had the entire test been rerun. This is particularly the case with the integration of several units, where it is too costly to go back to the lower levels even though a change has been made in the unit as a result of a detected interunit bug. Automation of testing and test design will allow the complete rerun to be performed. The net effect would be not only to substantially reduce the effort required to perform the level of testing being done today, but to significantly improve that level as well, at little or no additional human cost.

2.2.5 Functional Test Package

2.2.5.1 General

The functional test package is used to generate the core of a formal acceptance test. The point of view here, unlike that of the structural test package which is internal (based on the flow chart topology) and is heavily implementation dependent, the functional test is external, functionally oriented, and for the most part, is implementation independent. There is no single, comprehensive package that can be used to generate the numerous functional tests which comprise a complete system shakedown. There are, however, a number of areas in which the labor content of specifying and executing a comprehensive functional test can be significantly reduced.

2.2.5.2 Scope of Functional Testing

Functional testing, as distinct from unit testing and system testing, is epitomized when done by a formal acceptance test. Acceptance test plans and their execution, for a typical switching system require several man years of effort. A simple commercial switching system test plan could be devised in less than a man year. The total labor content (including vendor, programmers, evaluators, reviews, etc.), for the design, review, and execution of a formal acceptance test plan, for a large scale military system, is of the order of 10 to 15 man years. Often, this effort is so diffused among designers, evaluators, installers, or carried as part of design or monitoring, that it is difficult to pin the expenses down. In any case, it is a large item worthy of some investment to reduce labor costs.

2.2.5.3 Component Elements

2.2.5.3.1 Format Generator

No small part of message switch processing is concerned with checking and converting formats. There are the obvious header formats and routing indicator formats. However, there are other formats that are not usually recognized as such, but nevertheless require extensive programming. These include operator commands, traffic service commands, table generator input formats, retrieval requests and the like. Two cases must be considered in setting up the test for formats - valid and invalid cases. The system must pass and properly execute action on the receipt of valid messages and commands. Similarly, the system must reject and/or correct or properly dispose of errored formats. In all, the format tests (both valid and invalid) while being the easiest part of the test to write, are the most laborious and time-consuming. Almost half of the documentation and the executed tests consist of format related tests.

A format generator, given the formal syntactic rules that define a format, would generate character strings corresponding to every valid format combination, and character strings corresponding to the most likely invalid formats. Since the invalid formats are infinite in number, only a restricted set can be produced. Every single error should be generated. If the format generator is available, it should be possible to go on to the analysis of error pairs within the same format.

Most of these same arguments and comments hold for classmark generation and checking in circuit switching systems. For this reason, a classmark generator is recommended as well.

2.2.5.3.2 Traffic Generator

There are three generic conditions or kinds of traffic which must be generated in a test; test traffic, background traffic, and load.

Test traffic is used to test specific capabilities of the system. In general, test traffic is peculiar to the system and must be hand-crafted. It is a lot of traffic, but since the bulk of the effort is deciding what traffic is to be run, the opportunities for automation are minimal.

Background traffic is traffic which is kept running throughout the test. The intent of background traffic is to show that the system performs its functions while simultaneously running a background of representative traffic. This is not a difficult set of traffic to devise and would not benefit significantly from automation.

Load traffic is used to deliberately attempt to break the system down: or, at least, to test the system's defenses against higher than design loads. In general, this is a large amount of representative traffic, which should and could be automated. Running the same message or set of calls through is not safe. Each message or call must be self-identifying and unique: otherwise, lost traffic, garbled traffic, and the like will not be readily identified. The significant identifying data is usually carried in the text of the message or in the distinction of the call. The load traffic generator would produce coded sources of traffic in the desired quantities, in a form suitable for introduction into the switch.

2.2.5.3.3 Traffic Simulator

Much of the system cannot be tested without a simulator that is used to present valid and errored traffic to the system. Consider the problem of testing a channel coordination procedure. A pair of lines are cross-patched and traffic is introduced, say from a retrieval file or an intercept file. The message goes through properly, supposedly proving the validity of the procedure. All that this has proved is that the right hand of the procedure knew what the left hand of the procedure was doing. It does not prove the validity of the procedure, only its self-consistency.

As a step up in proving a channel coordination procedure, live testing might be used with an already proven user of that procedure. The messages go out in both directions and are received without error. We have proven the normal path of the procedure, but not the tougher and far larger set of error condition paths. To test these, errored traffic or errored responses must be simulated to the procedure. Since the other side of the interface is that of a working system, forcing it to create errored responses will require a patch - something that the operators of a working system are highly reluctant to do. A traffic simulator could do this job.

The problem of providing a proper load to a switching system has also always been a problem. Most often it has been done by some kind of cross-patching or loop-patching of inputs into outputs. While this creates a load, it is unrealistic. The load tends to be synchronized; this is a statistically harsh situation which often requires program patches to correct (decreasing the reality of the test). Input and output cannot be balanced as in a real situation, or else overflow will quickly result. Errored traffic cannot be sent or created.

For these reasons, a traffic simulator can be an effective tool in performing system functional testing. The simulator does not require additional or different hardware. It can typically be run in a minimal configuration of the same hardware being used to implement the switch. The simulator does not require the same elaborate accountability and protection features.

Furthermore, recovery from hardware faults is not necessary. The simulator, then becomes a software package to be implemented using the switching hardware.

2.2.5.3.4 Test Administration Package

No small part of the effort involved in producing a comprehensive, formal acceptance test, is the generation of a formal acceptance test plan. The test plan contains the complete documentation of the tests, indicating the nature of the test, a narrative description thereof, a complete specification of input traffic, operator actions, scenario components, etc., and a complete specification of every aspect of the system's behavior in response to the test, expected outputs, etc. Also included in the test plan are various message cross reference indexes, line indexes, test tables, call tables, etc. In all, the test plan document is of the order of 1500 to 3000 pages of text. Approximately 30% of the total manpower involved in creating this test is the administration and correction of the test plan document. Long before the formal test has been run, the test plan has been executed, piece by piece, in a series of dry runs. The test plan is as complex as any piece of software and must also be debugged. Most of the bugs that will be caught through the formal running of the test are caught as a result of designing the test and dry-running it. Errors fall into the following categories:

(1) Errors in the specification. The specification will be corrected, the programs changed, and the acceptance test plan modified.

(2) Errors and doctrinal misunderstandings in the operation of the system. The manuals will be clarified, the test plan narratives will be expanded.

(3) Program bugs. The bugs will be corrected, the test plan may be modified as a result.

(4) Test plan errors of understanding. That is, testing of a non-existent feature. The test plan will be modified.

(5) Doctrinal errors in the execution of the test plan. Test plan documentation will be revised.

(6) Test table bugs. Test tables will be modified, test plan documentation will be modified.

It is found that the test plan documentation is subject to as much, if not more, modification than is the system documentation. If that documentation were maintained on-line and could be edited by the originator of the change, much as programs are edited under the general documentation package (see below), a considerable amount of man power could be saved.

2.2.6 Model Builder Package

2.2.6.1 General

The continual development of new computer controlled communication systems, and the continual enhancement of existing systems implies, in addition to new code, new analyses of performance. This is an ongoing activity which is usually thoroughly integrated with the design and development activity. Comprehensive models of systems have not generally been done in the past. When done, they have generally been done from scratch for each new project, independent of the analytical effort that may have taken place on a similar or almost identical system based on the same machine. The problem is the same for the analytical area as it is for the software area. That is, the same barriers that exist to common software are operative in preventing common analytical efforts. Given modular software and given a solution to the software commonality problem, the possibility of common analysis emerges.

The thrust of the analytical approach proposed here, is to eliminate the bulk of it by creating a model building and maintenance facility, which is totally integrated into the software development activity so that common software will automatically result in common models for that software.

2.2.6.2 Use of the Model

An analytical model, if sufficiently comprehensive, is an indispensible management tool for deploying multiple systems based on the same hardware and software. Assume that a hardware/software package of the proper type has been created and the relatively mundane problem of doing site surveys and implementation of, say, a hundred sites must be performed. No two sites are alike. The traffic picture is different, the line structure is different, and the mix of operational requirements differ, even if the software is identical. Furthermore, it is not likely that all special features will be eliminated by fiat - it can be expected that certain minor differences from site to site, or from user agency to user agency, will exist.

The analytical problem to be faced is that of determining what resources are required to handle the traffic at each of a hundred sites. That is, how much core, how much mass memory, what channels, devices, etc. It is desirable to obtain a working configuration which neither over-buys nor is excessively restricted in terms of future growth. A typical analysis of the type required entails an effort of some two man years.

The individuals involved must be thoroughly familiar with the details of the hardware, the software and the application: in short, they must have been part of the original design team. This is the way it is done traditionally.

It is totally unrealistic to assume that it will be possible to obtain and maintain a staff sufficient to the purpose. Analytical automation is essential. That automation is most readily provided in the form of a comprehensive mathematical model, or rather, a library of model components which can be integrated, optimized, etc., before deployment so as to obtain an effectual system. This is the predominant use of the mathematical model.

The secondary use of the model is in the management of individual sites to determine the best way in which to meet the predicted or actual traffic growth. The traffic processed by the operational site is usually significantly different from the traffic predicted for it. Furthermore, as the system is used in the field, the original traffic predictions are seen to become less and less realistic. The site manager must also be able to plan his resource expansion requirements. The analytical talent and know how is not available to him, except through the use of a centrally administered comprehensive mathematical model facility. It is expected that each site will update its traffic projections at least on a semi-annual basis, and to obtain the resource requirements projections that often. Again, this would represent an incomprehensible burden if not automated.

The third use of the comprehensive model is in the development phase be it the original development or a subsequent enhancement. Its use is no less critical there - in fact it is there that the basic decisions on system behavior are made - if those decisions are wrong, or if a trade-off has been overlooked, or bypassed because it was too complex to evaluate manually, the resulting systems may all be operating at less than potential efficiency. An integrated, comprehensive mathematical model is an indispensible aid to development.

The final use of the comprehensive model is the role it can play in high level planning for a network or a communication agency. Changes in formats, changes in traffic patterns, changes in the scenarios for which the system was built, testing of hypothetical scenarios as a part of a larger scale military planning exercise, are all reflected in changes in the traffic and load which the systems will have to handle. While it is not practical to include a detailed mathematical model of a communications computer complex in a network model or in a higher level model, such as a war-games model, the mathematical model of the switch can be used to develop a summary behavorial mathematical model which could be incorporated into a larger scale model or simulation.

2.2.6.3 Building the Model

It is not the building of the overall model that is the problem but the modeling of its component parts. Most of the analytical effort involved in constructing a proper mathematical model can be automated and completely integrated into the design process. The primary tool for this is seen as an extension of the assembler/flow charter/test generator, resulting in a new package, the assembler/flow charter/test generator/model builder.

The topological outputs of the flow charter, when coupled with the information retained in the structural test generator, are the outputs required to construct a mathematical model of every subroutine, subprogram, etc., in the system. The assembly code allows precise timings to be made automatically. A specialized software program could be used to provide a complete algebraic model of any subroutine or program, automatically and without human intervention. Such a model would be an algebraic expression in the probabilities associated with the various decisions. These would be the components with which the mathematical models for the site would be constructed. The same kind of facilities that would be used to assemble the component subroutines into a working software package would, in parallel, be able to assemble the component algebraic models into a complete algebraic model of the system. An interactive system, such as GASM (see Volume III), expanded somewhat, and run in a non-interpretative mode, could be used to particularize the algebraic models. Human inputs would still be required for some of the probabilities, specification of traffic, etc. However, there are no fundamental problems in the creation of a model builder facility or the subsequent creation of particular models using that facility.

An expanded model manipulator performing the same kinds of things done by GASM does not represent any significant expansion.

The tie between the assembler/flow charter/test generator and the specialized software does not represent fundamental problems. Similarly, the tie between the specialized algebraic program and GASM is straightforward. It should also be pointed out that the algorithms required to find the shortest paths, etc., for the test generator would be part and parcel of the specialized program.

2.2.7 Target Machine Simulator

2.2.7.1 Reason for a Simulator

There are many advantages to doing initial testing, particularly at the unit level in a simulator of the target machine, rather than in a real target machine. A simulator can be integrated with performance monitoring and measurement software without introducing artifact. A simulator, running interpretatively, allows elaborate traces and traps to be implemented, which can be cumbersome or misleading in a real system. Most important, a simulator of the target machine is stratified in that one can operate at the simulation level with the assurance that there is no possible bug which will work in such a way as to deny the programmer the use of the debugging tools. In a real system, on the other hand, a bug could clobber the trace package, for example, making the finding of the bug difficult. The final and probably most important advantage of the simulator is that it can be incorporated into a time-sharing system, allowing multiple users to test programs simultaneously without fear of interaction.

2.2.7.2 Use of the Simulator

The simulator would be used for all initial assemblies and unit testing. If the simulator is large enough to incorporate the characteristics of the various peripheral devices, then testing could be done under simulation, up to the integration of a complete system. This would be very desirable. Successful running in the simulated mode does not guarantee that the system will run properly on the real test machine or the target machine. However, if it does not run successfully and pass all tests on the simulator, then it will surely not run on the real thing.

The simulator would also be used in conjunction with testing on the real system. In fact, a simulator access console would be available to the programmer right next to the real systems console during debugging. Assume that a bug has been detected. It is much easier to establish the conditions that led to the bug, to probe the cause, and to test possible fixes on a simulator, or with a simulator along side. For example, the act of probing on the real system can cause interference with the situation that exists in the real system; e.g., the bug is such that it affects or is affected or masked by the introduction of a trace or dump program. It would be easier to simulate the effects of the bug on the simulator where things can be tried over and over again without information loss, prior to doing the same thing on the real system.

2.2.8 Configuration Generator/Specifier

The configuration generator's primary function is to establish the values of assembly time parameters and tuning parameters required to particularize the system to the specified site. It is a SYSGEN package and, in principle, not new. This kind of package, which reduces much of the effort required to particularize a program to a site is most often done as an add-hoc thing, rarely designed from the ground up, and often totally lacking. There are many instances in which the particularization of the program parameters to a given site has been done by hand, resulting in a whole new set of bugs. While mechanical configuration generation will not eliminate all manual operations, it will, at least, assure that the particularization is complete and consistent.

The configuration generator is also needed on site. It must be used every time there has been a hardware change. It would also be used for line configuration changes and tuning parameter value changes that are required to best match the system to the traffic.

2.2.9 Table Generator

Switching systems, when properly designed, are table driven to the maximum feasible extent. The source tables may require from 20,000 to 300,000 statements. Every bit of the permanent tables must be specified. These include line parameter tables, classmark tables, routing tables, special delivery instructions, security data, priority information, accountability, and a host of internal parameter values. In some more advanced systems, table driven channel coordination procedures, table driven routing procedures, as well as other table driven functions are found, which are normally done in code.

The setting up and testing of the tables is no small part of the programming effort, though not usually done by the software developer. Most often, it is done by the user organization in conjunction with the software developer. In any event, specifying the tables is a programming activity. As an example of the extremes to which this can go, it is estimated that the labor required to set up the tables for the IBM-PARS reservation system is of the order of 40 man years. An expenditure of three to four man years for setting up the tables of a large switch is not unusual.

What is unusual is that we ask the table programmers to do the job in octal, or binary, or a haphazard mixture of octal, hex, binary, and ASCII. If lucky, the system will be provided with a table generator language comparable in terms of facilities such as listings and diagnostics to a relatively primitive assembler. Furthermore, the table generator source code, when a language has been devised, is unique to that system and bears no relation to any other system or the method used to introduce the tables. This is an abominable state of affairs.

Much of the source code of the tables could be made machine independent and be executed in a higher level source language without penalty at the object level. The tables should not be treated as if they are data, but should be treated for the code that they indeed are. What is needed then, is to define and develop a higher level table specification language with sufficient freedom to define the tables in the first place, and to specify the input formats where special formats must be devised.

The intention is to create a source language that would take care of the routine kind of things in a standard, site independent manner. The ability to do this would significantly reduce training costs for site personnel.

3.0 A NEW MODEL TOOL KIT FOR PROGRAM DEVELOPMENT

3.1 Software Perspective

If a truly homogeneous family of communications systems is desired, based as much as possible on common software, and a common architecture that is to be used over a wide range of applications, it is clear that a comprehensive support and development software package will have to be developed that is no smaller in scope to the kinds of software libraries developed for a commercial family of machines. By no means should this be interpreted to mean that a commercial software package is what is needed. If the above allegation is accepted, and the intention to reduce the total life-cycle cost of software development and maintenance to be associated with the CPS is pursued, it will be found that the basic software package, excluding the so-called application packages, exceed the engineering effort required for the hardware development. This should not be surprising since this has always been the case for commercial systems of comparable complexity.

Recognize that software costs are increasing, that engineering costs are continually being moved from hardware areas to software areas and, with it, basic software costs will similarly increase to overtake initial hardware engineering costs. While software costs cannot be eliminated, there is ample room for the significant reduction in labor content through the construction of modular software, the automation of many elements of program development, and through the provision of facilities comparable in scope, but specific to the CPS needs, that have long been enjoyed by the commercial programmer. Real-time systems programming may be difficult, but there is no need to make it any more so than it has to be.

3.2 General Features of a Comprehensive Software Development Operating System (COMSDOS)

Having seen the range of packages and software elements that exist or should be developed in support of a complete communication system design and operations activity, it is obvious that the integration and interworking of such a package requires something like an operating system. Furthermore, it is seen that much emphasis has been placed on doing things on-line via remote terminals. In other words, the operating system in question is a time-sharing operating system.

Such a system would have many of the features commonly found in a commercial time-sharing system. It would differ primarily in the fact that there would be a closer integration of the various component packages, elimination of much of the clumsiness now required to go from package to package, and the inclusion of a number of specialized file structures suitable to the restricted purposes of the software development effort.

Exactly what facilities are required, how they should be best integrated, how they should interface with the programmer, how they should be implemented, on what machine, etc., cannot be answered now. Experience has shown that many of the important requirements are recognized only after people start to use the system, by which time it is too late to go back and change things. A valid approach to solving this problem would be to build an initial version (admitedly inefficient) over an existing time-sharing operating system, such as that of the DEC PDP-10 or the HIS-6050. A test bed could be devised within which the efficiency of various approaches to communication system software development could be examined. Successful operation of the test system for a few projects would identify what features the dedicated COMSDOS should have. Many of the component elements could be constructed and evaluated on existing time-sharing systems as applied to any large scale real-time software development effort. This could be done without prejudice long before the CPS and its software package became available.

3.3 Component Elements

(1) Assembler, new model

(2) Flow charter

(3) Structural test generator

(4) Functional test package:

- Format generator
- Classmark generator
- Traffic generator
- Table generator
- Traffic simulator
- Test administration package

(5) Model builder/analyzer

(6) Target machine simulator

(7) Utility package

- Dumps

Converters

Traces

Editors

Test point

Loaders

(9) Performance monitors

(10) Hardware diagnostics

And an operating system/file management system that puts the whole thing on-line in an integrated manner so that it is reasonably easy to use.

3.4 Pre-Requisites

(1) Stable hardware

(2) Willingness to accept performance or functional compromises in the interest of reducing software costs.

(3) Funding of modular software as an effort of its own - rather than expecting it to come into being as a fallout of a specific project.

(4) A large number of configurations of comparable types over which the modular software and the software building tools can be amortized.

(5) Planning and construction of the entire package as a self-consistent entity rather than following the classical historical piece constructions of the past.

3.5 Conclusions

Software costs can be significantly reduced given stable hardware by increasing the amount of modular software. This implies an acknowledgement of the fact that modular software cannot be developed within the context of any one system but must be funded as a separate project whose objective is to create modular software to be used on all projects.

Software costs can be significantly decreased by taking assembly language programming out of the dark ages by providing reasonable tools, which have, for the most part, been denied to the assembler programmer.

3.6 Recommendations

(1) Establish a study to consider, in further detail, what the mix of software development tools should be. The objective would be to provide sufficient detail to allow an implementation on a trial basis.

(2) Implement the major elements of the programming tool kit as interpretative programs on a time-sharing system and test it in use in the development of a small, but representative communication system. Revise the specifications for the final package in the light of what has been learned. This should be done in parallel with the CPS hardware studies so that time is not lost.

   a. Consideration should be given to implementing portions of the package which are theoretically feasible, relatively independent of the entire package, but which still require operational testing.

(3) Integration of the package and rewriting thereof for implementation in the CPS computer for a CPS software effort - e.g., creation of the basic modular software to be used in the CPS based system.

4.0 <u>SYSTEM GENERATION</u>

The capability to identify, select, and configure Circuit Switch and Message Switch software across varied physical and functional arrangements was investigated. Recognition of the desirability and usefulness is easily obtained, but can instantly be matched by awareness of the complexity of such an undertaking.

It was, therefore, the intent of this investigation to uncover and itemize basic criteria which would have to be addressed during such an endeavor. An attempt is made to uncover the salient problems which would be encountered, and to arrive at some corresponding solutions. Detailed implementation characteristics and practices are left to the planning and construction phase of the System Generation facility.

While the purpose of this investigation is directly related to the construction of Circuit and Message Switching systems, it is not envisaged that the concepts presented here preclude their use elsewhere. This premise is made based upon the assumption that many large systems are constructed in a manner similar to the systems in question, and that they possess the same requirements for reconfiguration of their parts to provide a working subset.

It will be beneficial at this point, if a brief definition is given for System Generation, as it is used here.

The concept of System Generation classically applies to the software system only. It is considered to be a tool with which preformed software programs, each of which address small sections of switching logic, are selected and grouped together to provide the total software needed for a particular switching environment. Specifications for that environment are presented to the System Generation facility, which in turn, accesses software programs which will satisfy the stated objective. The resultant delivery then, is a system program, contained on some media (tape, disc, etc.), which can be loaded into the target machine and operated.

An extension of this definition introduces the hardware modules into the process. This extension is made in order to accommodate various hardware system configurations which could be applied in the solution of a given switching requirement.

The System Generation concept presupposes certain system characteristics, among which are:

(1) The software constructed is modularly oriented such that individual programs can be logically and physically grouped.

(2) Functions to be performed can be explicitly stated.

(3) Coorelation between functional requirements and program module(s) can be developed and maintained.

(4) Equipment configurations encompass the entire spectrum of resources which need to be applied for any configuration demanded.

Although System Generation, in this context, does not consider the system data associated with all switching systems, it is a matter of importance and requires special study.

4.1 SOFTWARE DESIGN

Prime importance must be placed upon the software developed for use in a System Generation procedure, so that it is suitably constructed. This is no small feat. At the onset of the software development, expected results must be outlined with achievable goals in order to meet the intended result. Otherwise, the software modules which functionally constitute the individual components will not necessarily be developed to insure their unique callout and concatenation capabilities with other module members.

An example can serve to illustrate the point. Assume that one of the goals to be accomplished is the construction of a series of software modules which provide "front-end" supervision processing for the main body of the call process work. Regarding a circuit switch application, therefore, several kinds of capabilities would need to be built into the software scanning logic. These would include all possible termination types expected in the system, frequency rates which need to be maintained, signal validation procedures per type, error detection requirements, data delivery to intermediate processing modules, and program interface criteria between the scanning logic and other system modules.

Given that all of the above conditions and implementation details can be provided, a problem can still evolve which prevents the desired result. For instance, one can imagine that the software designer develops the scanning logic to include all functional capabilities as one large program. They are, therefore, integrated together in such a manner that the entire program must be used to exercise the capabilities for only a selected set of termination types. This defeats the intended result of the process.

It would seem logical that the scanning logic should therefore be parsed into small modules, each of which provide, say, process logic for one kind of terminal specified. These modules would then possess the capability of being linked together to comprise the desired scanning logic. Unnecessary modules would be omitted. The System Generation personnel would then need to specifiy each kind of terminal to be dealt with, timing characteristics and numbers of each such terminations.

The following sections deal with critical areas which must be considered during the entire software development process, leading towards a comprehensive System Generation facility.

4.1.1 Software Development Languages

The use of language here is concerned with the development tools available for application software, rather than specialized languages which assist in the System Generation function itself.

To the extent that languages play any role in the process, and are not themselves transparent, requires some investigation into the constructs of the resultant machine code. This is necessary only if more than one language is used, and in particular if the programs generated are independently produced.

It may serve a useful purpose in exploring why this would be attempted; that is, why would one choose to develop various portions of the applications software using different languages. The answer is dependent upon the kinds of languages made available for the processor, and the kinds of tasks which the operating software will be attempting to do.

It can be imagined that a switching language, peculiar to the application tasks is available. It may also be imagined that other languages, tailored to other specific tasks and optimized for those tasks is available. These might include a data base manipulation language, a statistical language which retrieves system data and massages it in some manner, and an expanded assembler used for special coding requirements. All of these language tools might be made available to the software designer to effectively and efficiently perform his task.

Given the diverse languages which could therefore be available, it is important to consider the interrelationship of the resulting code produced among the systems. It is expected that the definition and access of data items in all languages would be compatible. This is of concern since a logical relation between the languages should exist at least at the data level. However, it may not be convenient to retain consistency of data specifications among the languages used. A problem in interfacing programs generated from different languages could then result.

An example can illustrate the point. Assume that the application program is written in a high level language format. The data used and processed would then be defined at a high level also. Certain conventions and restrictions would than be followed in using the data. Assume also that a special language is developed for the data reduction of the information acquired by the application programs. This language might be statistically oriented, requiring data items and fields to follow some specific format conventions. These conventions may not necessarily coincide with the conventions required by the application program processes.

The System Generation procedure would have to take incompatible data field conventions into account when programs generated from both language systems are combined. It is logical to assume that this should be handled by some automated process, transparent to the user of the System Generation procedure. One such automated process could be the envoking of a data transformation routine, which prepares the information for processing prior to the actual processing activity.

It is also possible to imagine the interfacing of programs generated by diverse languages at the code level. Programs interfacing would, therefore, take place by transferring control from one program to another through some calling sequence. Here again, the conventions developed may be inconsistent between the languages, perhaps in the parameter handling area. It would seem logical then, to provide for these inconsistencies within the System Generation procedure as was suggested earlier. In this case, however, it might be better to adjust the parameter handling process of one of the languages in question, rather than to introduce data transformation procedures which necessitate gross expenditures of time and core.

The solution to these kinds of situations would have to be addressed during the design of the System Generation software, taking into account the idiosyncrasies of the languages which support the applications software.

4.1.2 Program Modularity

It will be of great importance in provisioning for the System Generation eventuality that the programs be composed of small pieces of logic which can be grouped together to form a specifiable system function. Otherwise, tailored software configurations cannot be created, which is the intent of the exercise.

The modules of software should then reflect some criteria by which they can be judged in meeting the objective. It may be difficult in defining such module constraints, to accommodate all possible implementations, without imposing severe limitations on the design. However, the result is worth achieving and should therefore be pursued.

Fortunately, a design technique has evolved which purports to encompass the attributes which are necessary for modularized software. This concept is included in what is known as "Structured Programming". A section of this report deals with the details of this approach. It is sufficient now only to point out that a method does exist, and can be used to support the System Generation objective.

Of major concern then, is to define the level of modularity with which systems should be produced. It is difficult to imagine a truly effective System Generation procedure which has imposed upon it program modules which were designed independent of any such considerations. The result would be, when considering historical software generation techniques, a group of very large program modules, each of which provide many special system functions intertwined in a manner which disallowed their segmentation. At best, it could be expected that these large modules would have built into them software "switches" which enabled or disabled certain sections or features. But the total code would need to be carried into any system which required even a small portion of the module logic.

The idea will be to parse the software functional requirements into a group of smaller functions. These smaller functions may in turn be parsed into several functions themselves, etc.

Depending upon the overall size and top level structure of the software system, the parsing activity may evolve into a myriad of very small routines and subroutines. These small modules will contain only a fraction of any particular function which the system specification demands. However, the modules are small and manageable, which was one of the results to be achieved.

The following remarks must be prefaced with a definition of functional software and program organization.

When viewing a major software implementation, it is necessary to consider the functions which the system is expected to perform. This is because software has been designed to perform efficiently these desired functions with respect to both time and memory utilization. Therefore, in a Circuit Switch system, single streams of logic to encompass each type of call placement (seven digit local, seven digit extended area service, two digit abbreviated dialing, etc.), are not found. Rather, the software would be constructed in many major pieces, each of which provides some portion of many call types such as a portion which handles supervision signaling, another which handles route translation, another which handles digit collection, etc. In this manner, both execution efficiency and minimal memory utilization is achieved.

This organization leads to the understanding that these programs are composed of many small functions, each of which contributes to some portion of one or more system functions. Because of this, some of the modules will be interrelated and dependent upon one another, while others will not.

Returning now to the main topic. In viewing any partic- program the question can be asked, how is it known which modules need to be grouped together to form any one of the functions which the System Generation personnel may call out? A particular function may need to consist of from ten to one hundred of these small modules.

Two solutions come to mind. There undoubtedly are others.

First, the identity of each function which the system was capable of handling would be defined. This information is available since the software system was based upon known functions prior to the parsing activity. Then the identity of each module which assisted in providing the specified function would be defined, pulling them all together physically, so that each function was uniquely grouped. Each module still maintains its own separate identity, although it is physically grouped with others.

This may turn out to be the most viable approach. There are some drawbacks, however, the most serious of which is module redundancy. As each function was being identified as requiring several modules grouped together, these modules were separately lifted and placed with other modules to form the whole function. Each function of the system followed the same practice, independent of one another. It may evolve that the same module was lifted several times to form a part of several functions. This means that the same logic, if all functions are required for some configuration, could exist in memory many times over.

This is not necessarily a poor practice. The extent of these occurrences would have to be analyzed in conjunction with available memory space to properly judge the concept.

There is another alternative. It would be possible during the correlation between functions and modules, to "mark" each module for inclusion into each function that it serves. No physical configuration would occur during this process. Rather, a function identifier mark would be included with each module as an information adapter. This information could then be used during the system generation process to envoke each module as it is required. But the module would be envoked only once per system configuration, so that multiple copies of identical logic would not be retained in memory.

To name one drawback to this approach, it can readily be seen that module re-entrant capability might have to be introduced, in which case, more complicated software would result. This is not necessarily the case, however, since the structure of the software should dictate whether this need exists. With a thorough knowledge of the system operation, particularly in scheduling criteria, it might not be required to develop re-entrant routines. In any event, these kinds of capabilities should rightly be addressed in programming conventions and not here.

4.1.3 Linking Arrangements

In historical perspective, linking of programs is performed by a system loader, which operates on the target machine. The loader has the capability to group individual programs together and provide for their interrelationships. This requirement may still be required to some degree as it typically exists today. However, it is felt that much of this activity is required within the System Generation process itself, and to a much lesser extent on the system processor. Relocation processes of course, may still be preferable at system load time.

The linking process will have to concern itself with some module acquisition method. It may turn out that this task is one of the larger efforts in the construction of the System Generation procedure. Some of the subtasks which can be envisaged are:

(1) Identification of all modules within the system.

(2) Correlation of system functional requirements with individual module elements.

(3) Development of definitive procedures by which the modules are grouped together.

The above three subtasks are highlighted because of their expected major importance. There are undoubtedly others which deserve attention. A discussion of the identified tasks follows.

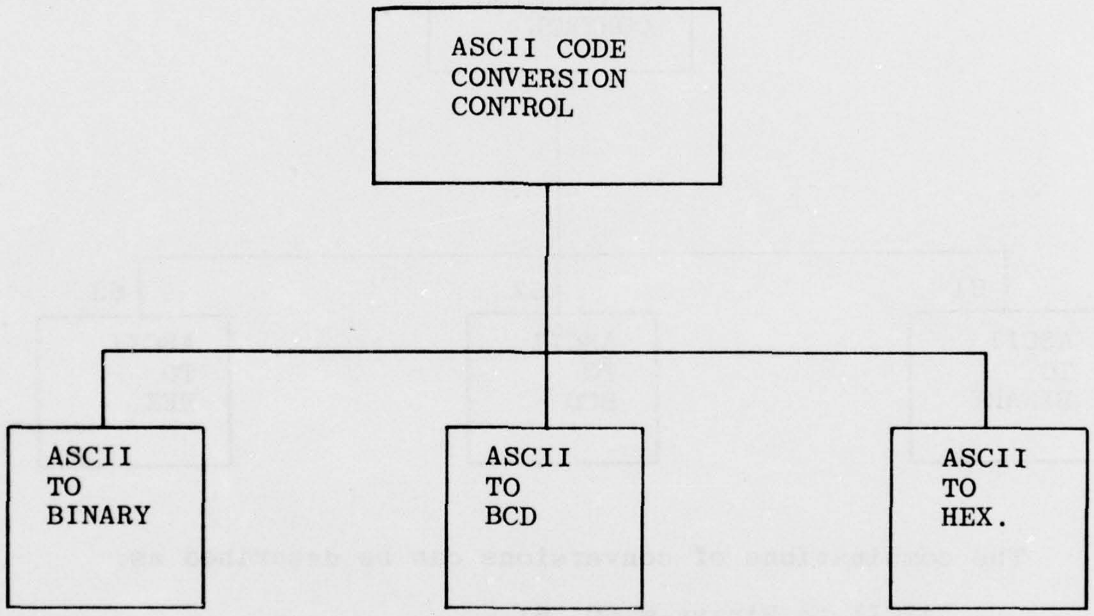
4.1.3.1 Identification

For a very large Message Switch or Circuit Switch, the number of modules will total a thousand or more. If the definition of a module is extracted from the structured programming method, then a physical limitation is placed upon a module which prevents its size from exceeding one printed output page of a program listing. Circuit Switch programs in the medium to large size, are currently within the 1000 to 3000 page printout range. These are primarily assembly language oriented. Message Switches produce even more code, again at the assembly level, when all off-line software is introduced.

It can readily be seen that the identification of modules will be no small effort. Some technique could be developed which automated this identification process. This would not be required, except that for very large systems the manual labor required could be excessive. The concept is important, however, and not the implementation criteria at this point. What is needed then is some unique identifier which can be associated with each module.

The next area which needs identification consideration regards the collection of modules which are dependent upon one another. This is not system functional dependency, but rather logical dependency among functions which have been segmented to accommodate rules inherent in Structured Programming.

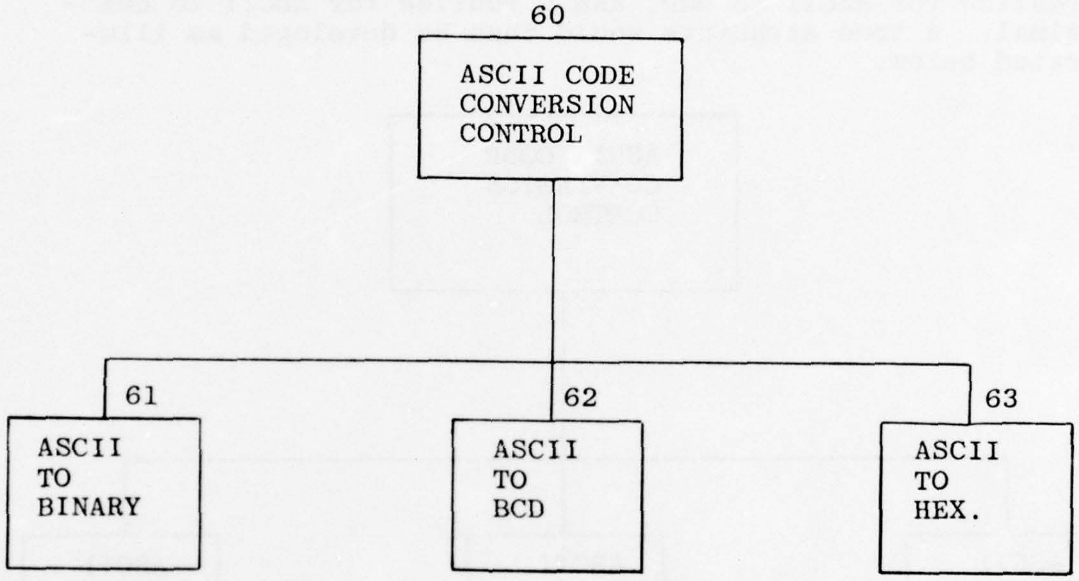
An example will best serve this definition. Suppose that as part of some process, a need exists to provide a data transformation from ASCII to one of several code formats, and that according to our knowledge of the system, it is expected that only a subset of these transforms need to be accommodated on any given system. As a software designer, parsing of the logic into several different logical and physical segments would be performed. For instance, a routine to convert from ASCII to binary, a routine for ASCII to BCD, and a routine for ASCII to hexidecimal. A tree structure would then be developed as illustrated below.



This ASCII code conversion block contains control logic which ascertains, according to some input it receives, to which transformation module control should be directed. The designer would then have to indicate that each of the three transformation modules required the conversion control module for its operation. Therefore, there would be specified three distinct groups of transformation schemes which could be selected.

In this context, two additional points can be made. The conversion control module would carry with it a name which was common to all three transformation schemes. The logic of selected modules would be built in such a manner to recognize multiple requests for the control module, and insert that module only once. Of course, the capability should exist which disabled this multiple selection and allowed redundant copies of the control module to exist. This is warranted on some occasions and a means to allow such duplication should be provided.

The identification of the modules within the system is important since they will be used to organize the software. The code conversation example would then be modularly identified as depicted below.



The combinations of conversions can be described as:

ASCII to Binary	= 60, 61
ASCII to BCD	= 60, 62
ASCII to Hex.	= 60, 63

Further refinement of module identification will be discussed in paragraph 4.1.3.3.3.

The second point is that some procedure would have to be developed which recognized, in the control module itself, requests for any conversion types which had not been specified. It would theoretically be possible for the control module to receive an input to transform ASCII to BCD, but the BCD module had not been specified. This would be an error condition which should not cause catastrophic situations to occur, if that event took place.

The third and final area which requires identification regards the system functions to support any possible capability which the system was intended to handle. This information will usually be contained within the system specification. The individual capabilities will have to be uniquely recorded, so that they may be envoked individually. For instance, a list for conferencing capabilities of a Circuit Switch would contain:

1.0 CONFERENCE

- 1.1 BROADCAST
  - 1.1.1 Pre-emptable
  - 1.1.2 Non-Pre-emptable
- 1.2 PROGRESSIVE
  - 1.2.1 Pre-emptable
  - 1.2.2 Non-pre-emptable
- 1.3 PRE-SET
  - 1.3.1 Pre-emptable
  - 1.3.2 Non-pre-emptable
- 1.4 MEET-ME
  - 1.4.1 Pre-emptable
  - 1.4.2 Non-pre-emptable

Each feature and capability contained within the system would have to be covered in detail similar to that outlined above.

4.1.3.2 Correlation

The next sequential process which would occur concerns the relationship between system capabilities and module utilization. The system designer will have cognizance of these relationships, and can logically be expected to provide this correlation. The task will be large and time consuming, but crucial to the System Generation process. It is not apparent that any automated scheme can be used here, although some technique might conceivably be devised to do so.

The correlation is envisaged to be as follows, again using the example on conferencing.

Feature 1

1.0 CONFERENCE; 1.1 BROADCAST; 1.1.2 Non-Pre-emptable= Modules 01, 02, 04, 51, 52, 53, 60, 61, 62, 78

Feature 2

1.0 CONFERENCE; 1.2 PROGRESSIVE; 1.2.1 Pre-emptable = Modules 01, 02, 03, 51, 52, 53, 60, 62, 63, 77, 79

The relationship depicted above represents an input to the System Generation process which will be used when configuration specifications are developed. Feature 1 indicates that for the feature for non-pre-emptable broadcast conference, eleven modules are needed. Modules are identified as 01, 02, etc. For a progressive conference which is pre-emptable, a list of the necessary modules are given. Notice that the code conversion modules outlined in the past section to construct these correlations have been used.

This information would be entered into a table, accessible by the System Generation logic, when either of these features capabilities were required. The process of selection during System Generation operation would be automatic.

The correlation process is considered in greater detail in paragraph 4.1.3.3.4.

4.1.3.3 Procedures

The immediately preceding sections concerning module and function identification and correlation have provided a preliminary method by which the software system can be viewed, constructed, and grouped. This section addresses the ordering of the modules themselves, relative to their hierarchial structure and execution sequence. This is a vital portion of the linking process. The modules must not be arranged in random, but in a predetermined order. That order requires specification.

Before examining this subject in detail, a digression is needed for background information. The overall structure of a software system should be viewed, in terms of how it is put together, and the interrelationship of its parts. A Circuit Switch structure is used for the example, highlighting these areas which deal with the call processing logic.

4.1.3.3.1 Background

In general, the basic elements of the Circuit Switch include the following distinct sections.

(1) Scanning Logic - The Supervision Signaling Function, which consists of that portion of the system which provides for the supervision of terminations attached to the system.

(2) Register Logic - The Address Signaling Function, which consists of the portion of the system which provides for the dialed digits reception, and causes the outpulsing of digits to distant switchboards. This logic would also receive digits incoming from other switchboards.

(3) Matrix Logic - The Matrix Control Function, which consists of the software which interfaces with the matrix to cause path connection and disconnection operations, and tone injections in certain systems.

(4) Translation Logic - Part of the Call Processing Function, which provides the section of the system which performs the analysis of address digit information, and translates that information into subscriber termination addresses or into a trunk group selection for calls progressing outward to distant switchboards.

(5) Non-Register Processing - Also part of the Call Processing Function, which provides the various call processing tasks which are performed prior to or after the dialing and translation phase of the call. This includes initial off-hook processing, classmark or feature privilege checking allowed per subscriber, removal of ring and ringback and associated timing, and busy tone control.

The above overview of the process does not totally present the entire picture, however. To belabor the definition a bit longer, the participation of each section must be considered during the establishment of a call.

The scanning logic operates periodically to determine on-hook/off-hook signals, among others. An off-hook signal is then initially handled by this logic, which delivers information to the non-register processing segment. In this process, the terminal which is requesting service is compared with pre-stored data, such as type of register required, direct access capability, etc.

If the data indicates that a register is needed, an idle unit is selected. The connection of the terminal to a register is handled by the matrix logic section. Thereafter, dial tone is applied and the register logic accepts digits as they are dialed. These digits are then processed by the translation logic which determines where the call should be terminated; i.e., another local subscriber termination or a particular outgoing trunk.

Assuming that a local subscriber is required, the connection is made between the two local subscribers, following disconnection from the register. Again, the matrix logic is envoked for this purpose. Thereafter, ring and ringback tones are sent to the respective terminations, and the call handling is then turned to the non-register logic section.

When off-hook is detected from the called subscriber, the non-register logic removes ring and ringback and the subscribers can begin conversation. From that point onward, the scanning logic is directed to report on-hook conditions from either subscriber, and upon that condition, a sequence would then be undertaken to disconnect the matrix path.

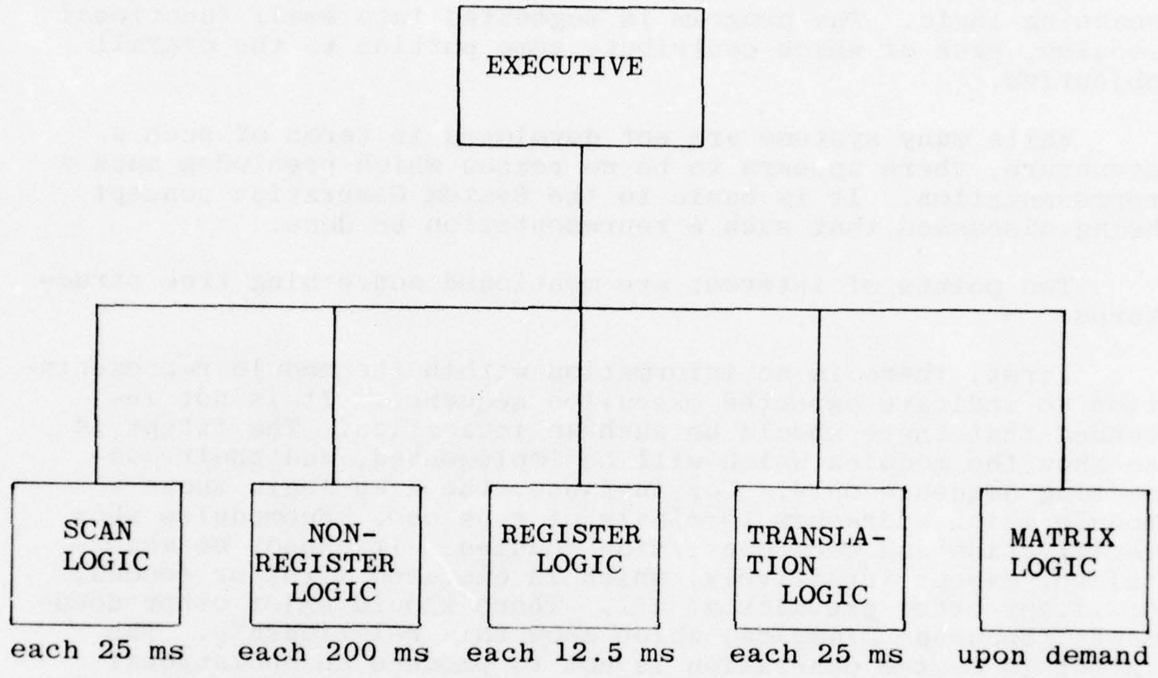
As can be seen from the above call procedure, different areas of the software logic are directed into execution to handle different portions of the call sequence. A similar situation will occur when special call features, such as conferencing, pre-emption, etc., are required. Most Circuit Switch software is designed in this manner, or variations on the same theme, because different portions of the call establishment require different timing considerations. Also because the system must accommodate many calls, simultaneously, each is likely to be in some different state of completion at any given time. The resources of the processor cannot then be totally dedicated to any particular call from start to finish.

A block structure of these major software programs is shown in Figure VI-1-1, with execution times associated with each block.

4.1.3.3.2 Structure

The functions required within Circuit Switch software as outlined in the previous section need to be considered in more detail. It is consistent with current practices of software design to view the software as a "Tree Structure". The beginning of such a structure was outlined in paragraph 4.1.3.3.1, where the major call processing modules were depicted. That representation can be expanded further.

Consider the scanning logic mentioned earlier. This program logic may consume 1 to 2 thousand instructions. It would, therefore, according to the rules of module size, consist of 20 to 40 small modules. Each module is interconnected in some predetermined manner with other modules to provide the desired program sequence.



MAJOR SOFTWARE ELEMENTS

FIGURE VI-1-1

An example of a portion of the scanning logic could be shown in tree structure as depicted in Figure VI-1-2.

The tree structure depicted shows some of the detail of the scanning logic. The program is segmented into small functional modules, each of which contribute some portion to the overall objective.

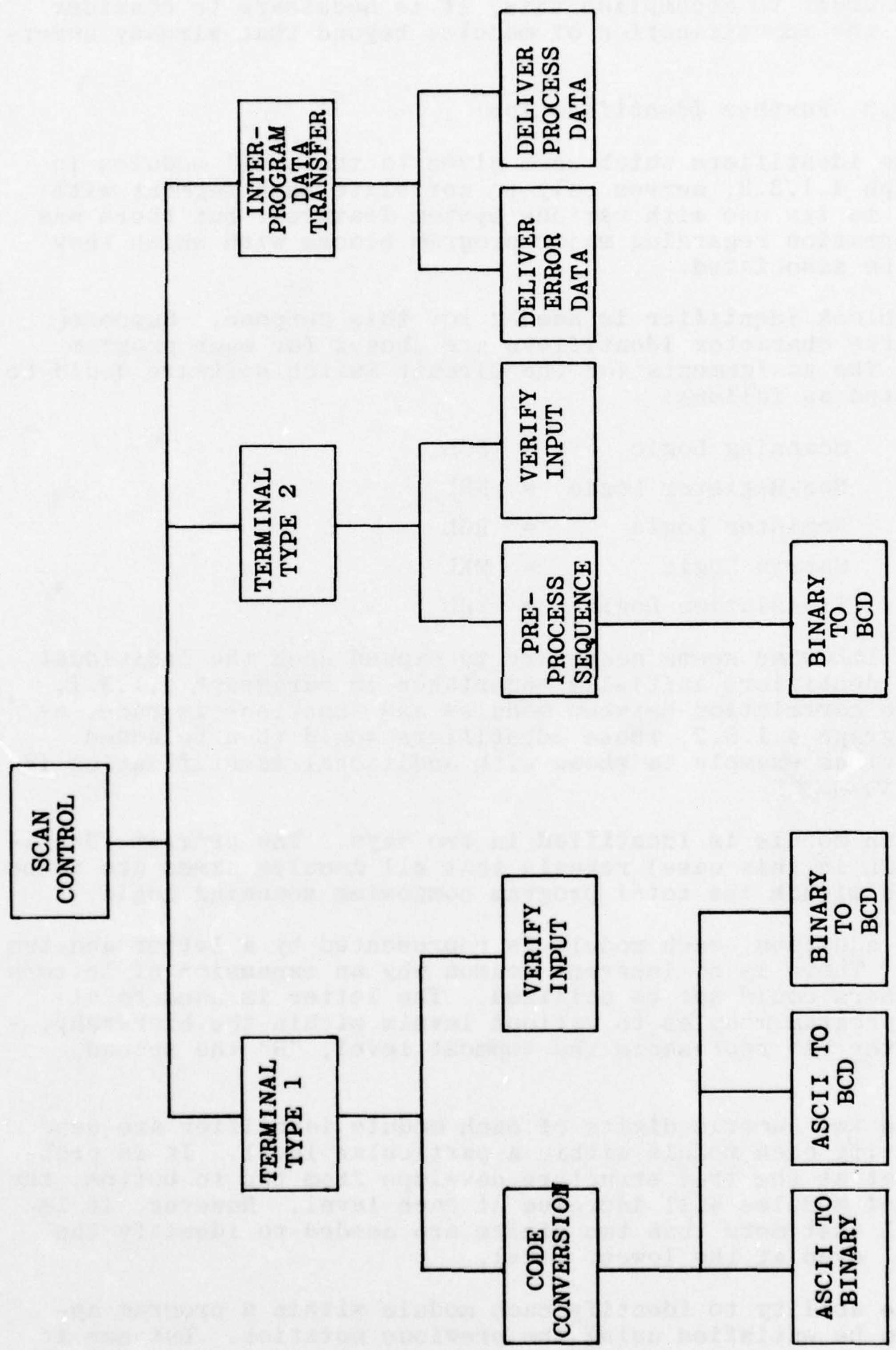
While many systems are not developed in terms of such a structure, there appears to be no reason which precludes such a representation. It is basic to the System Generation concept being discussed that such a representation be done.

Two points of interest are mentioned concerning tree structures.

First, there is no information within the module representation to indicate expected execution sequence. It is not intended that there should be such an indication. The intent is to show the modules which will be implemented, and their connecting sequence only. For instance, the scan logic shows a module which addresses terminals of type one. Submodules show verification and code conversion modules. It cannot be ascertained, except intuitively, which is executed first or second, or if any order prevails at all. There should exist other documents (Sequence Diagrams) which show this relationship. The intent in System Generation is not to produce an operational sequence knowledge, but rather to catalogue the components of the system and organize them into some workable configuration.

Second, for some branches of the structure, a block may be shown which appears to be identical to those which exist in other branches. (Branch is defined as a collection of modules which when taken together, constitute an identifiable system function). These branches, in fact, may contain some of the identical logic, duplicated or, on the other hand, the logic may not be duplicated. Again, the intent here is to provide for the tree structure to reflect logic in each area which is required, without showing interconnections to prevent duplication. As an example, the scanning logic shown in this section reveals a module called "Binary to BCD" which is repeated twice. It is necessary that this module be made part of both the Type 1 and Type 2 processing. Instructions concerning the requirement for duplication must be given to the System Generation process in order to handle it appropriately.

Returning to the main topic now, it can be seen that some method must be employed which instructs the System Generation facility according to the manner in which the modules should be arranged. It would have to know which modules should be grouped within the scanning logic, which into the non-register logic, etc., and how to arrange the modules within each section.



SCANNING LOGIC STRUCTURE

FIGURE VI-1-2

In order to accomplish this, it is necessary to consider further the identification of modules beyond that already undertaken.

4.1.3.3.3 Further Identification

The identifiers which were given to the small modules in paragraph 4.1.3.2, serves only to correlate each segment with respect to its use with various system features, but there was no information regarding major program blocks with which they should be associated.

A block identifier is needed for this purpose. Suppose that three character identifiers are chosen for each program block. The assignments for the Circuit Switch software could be designated as follows:

Scanning Logic	=	SCL
Non-Register Logic	=	NRL
Register Logic	=	RGL
Matrix Logic	=	MXL
Translation Logic	=	TRL

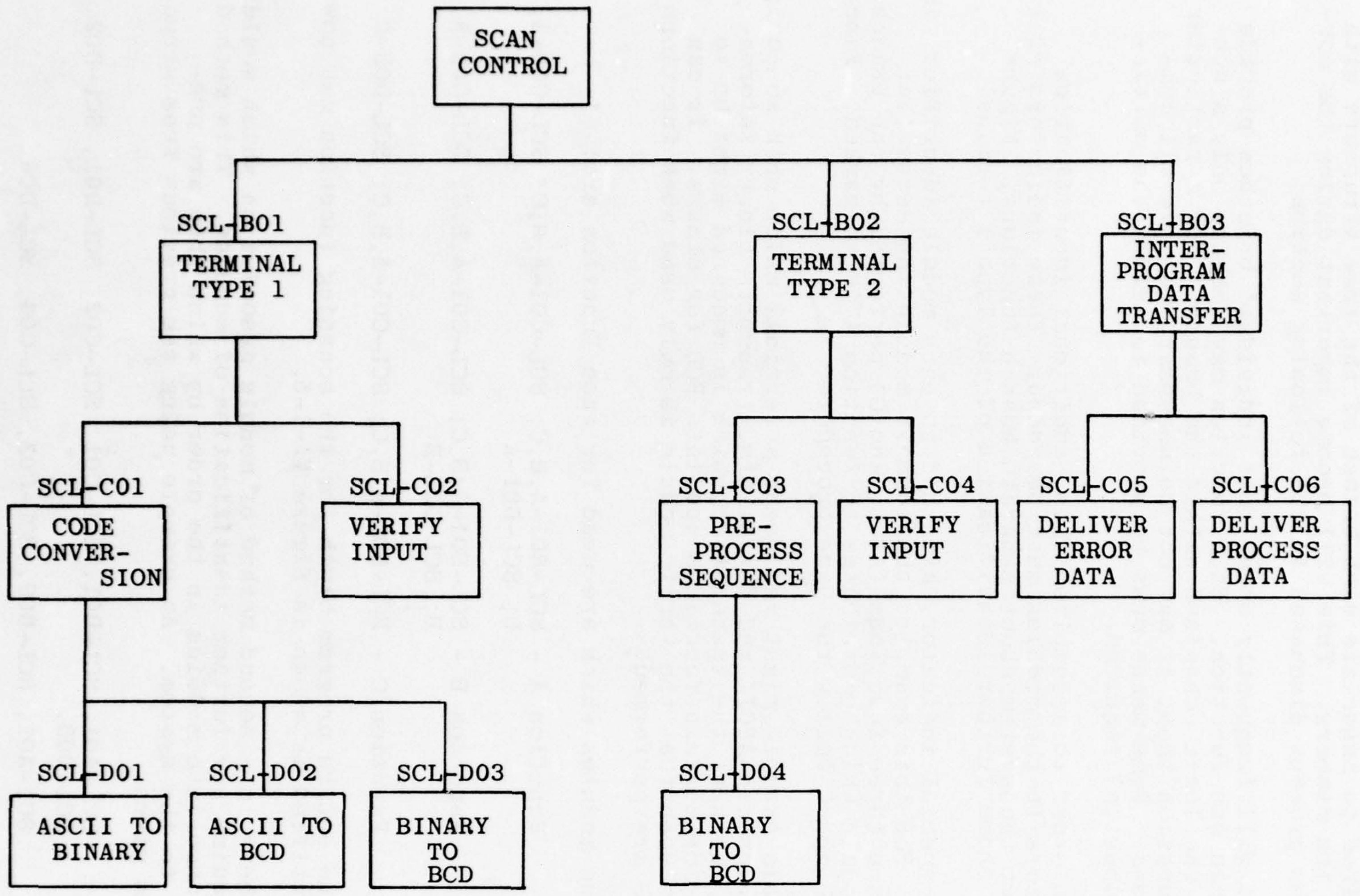
It likewise seems necessary to expand upon the individual module identifiers initially undertaken in paragraph 4.1.3.1. When the correlation between modules and functions is made, as in paragraph 4.1.3.2, these identifiers would then be added. The previous example is shown with additional identification in Figure VI-1-3.

Each module is identified in two ways. The program identifier (SCL in this case) reveals that all modules names are to be included within the total program composing scanning logic.

In addition, each module is represented by a letter and two digits. There is no inherent reason why an expansion of letters and numbers could not be utilized. The letter is used to allocate program modules to various levels within the hierarchy. The letter "A" represents the topmost level, "B" the second, etc.

The two numeric digits of each module identifier are used to identify each module within a particular level. It is probable that as the tree structure develops from top to bottom, the number of modules will increase at each level. However, it is doubtful that more than two digits are needed to identify the modules, even at the lowest level.

The ability to identify each module within a program appears to be satisfied using the previous notation. But now it



DETAILED SCANNING LOGIC BLOCK

FIGURE VI-1-3

is desired to associate each branch of the tree structure with all of its members. This will become important during the correlation process discussed in the following section.

It will frequently occur that individual branches provide more than one function. Each function may require only a subset of the logic contained within the branch. For a particular configuration then, it may not be necessary to have all combinations. Some means must be provided to select the particular subset(s) required.

In order to accomplish this, additional identification indicators to the modules must be added. These indicators will instruct the system about separate branch functions. Figure VI-1-4 shows further identification of the Type 1 terminal.

A special indicator, attached to each module identifier, is shown. For this example, the topmost module indicates that there are three functions (A, B, and C) performed by the branch. Each module which contributes to function A is so marked. Functions B and C follow the same procedure.

This example might represent a terminal which when so equipped, transmits ASCII and for others, transmits binary information. Further, the translation which is required might be to BINARY for certain functions and into BCD for others. It can also be seen that the verify module is only used when functions A and B are performed.

The modules which are used for each function are:

Function A - SCL-B01-A,B,C; SCL-C01-A,B,C; SCL-C02-A,B; SCL-D01-A

Function B - SCL-B01-A,B,C; SCL-C01-A,B,C; SCL-C02-A,B; SCL-D02-B

Function C - SCL-B01-A,B,C; SCL-C01-A,B,C; SCL-D03-C

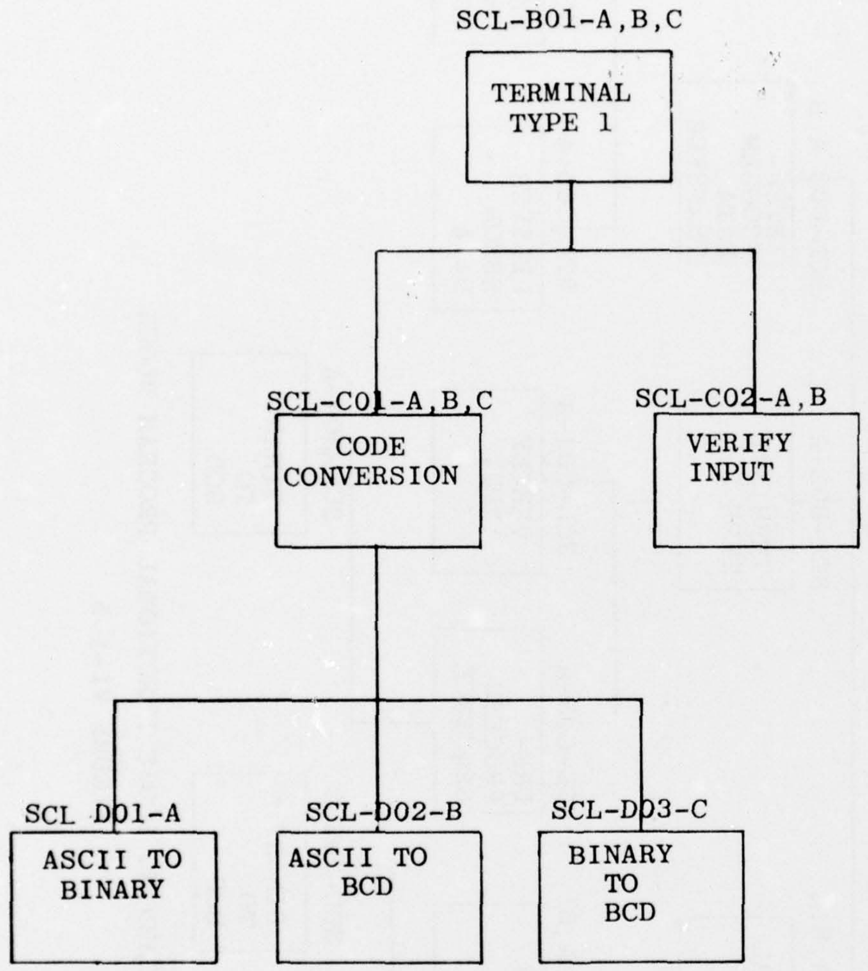
The entire program block for the scanning function can now be identified as shown in Figure VI-1-5.

There is a second method of module association which would not require any further identification of modules. This method would associate modules in the order by which they are presented to the system. An example using the previous tree structure is given.

SCL-A01, SCL-B01, SCL-C01, SCL-C02, SCL-D01, SCL-D02, SCL-D03.

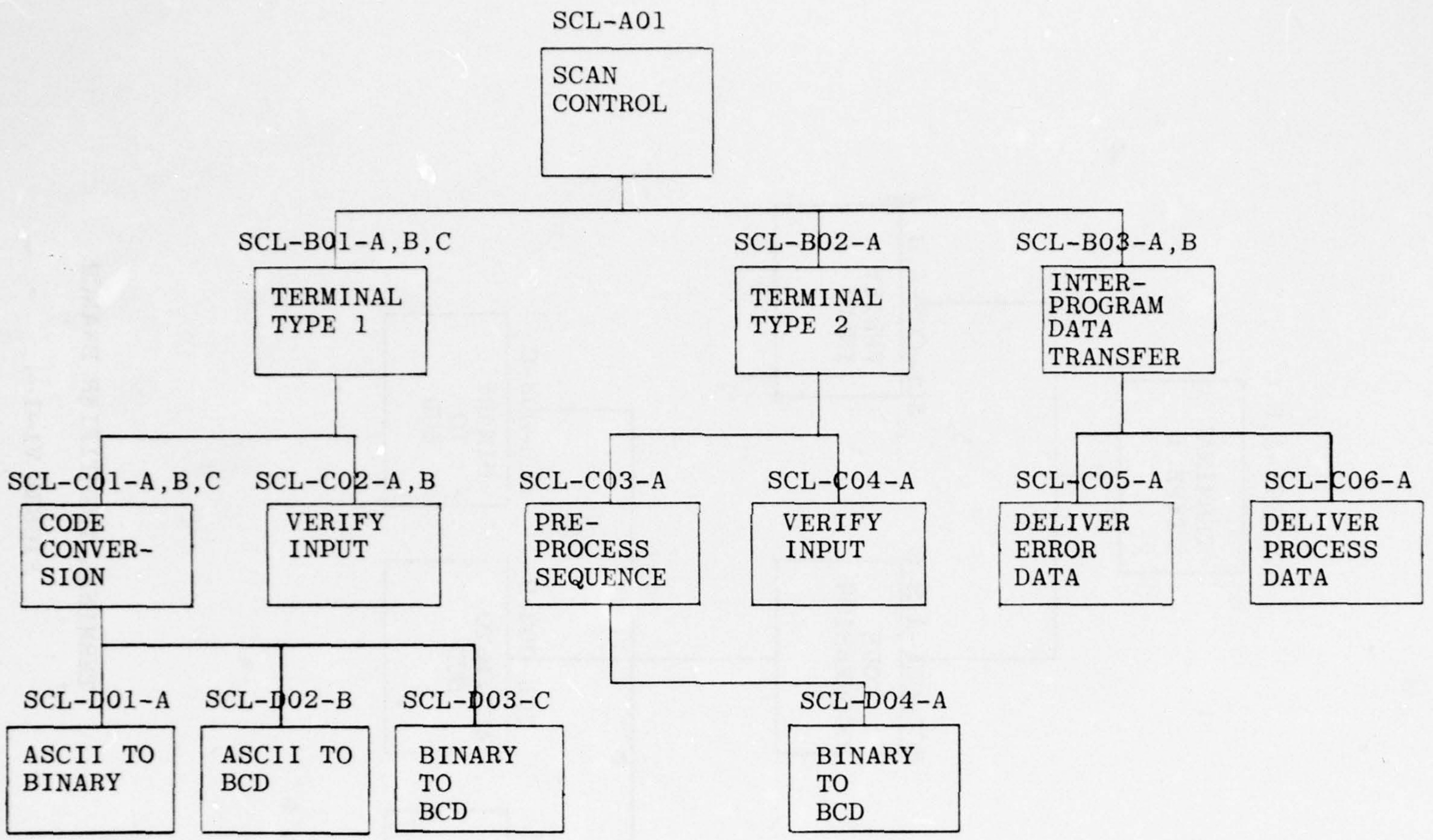
SCL-A01, SCL-B02, SCL-C03, SCL-C04, SCL-D04

SCL-A01, SCL-B03, SCL-C05, SCL-C06



TERMINAL IDENTIFIER BRANCH

FIGURE VI-1-4



DETAILED SCANNING LOGIC FUNCTIONAL PROGRAM BLOCK

FIGURE VI-1-5

This method is simpler because it does not require additional identifiers to interrelate the modules. However, it suffers in completeness because only main branches may be so associated, and lower level branches may not be specified. The lack of this capability may not incur any serious problems, depending upon the eventual use which is made of these branch identifiers.

For the purpose of the ongoing discussion, however, the first method outlined will be used.

4.1.3.3.4 Further Correlation

At this time, it has been possible to identify totally each module individually, associate all modules which belong to the same branch, identify modules within a branch which perform separate functions, and identify them with respect to major program blocks. The next step in the System Generation process involves the correlation between modules and functions. This is an expansion of what was begun in paragraph 4.1.3.2.

The need exists to identify every possible function which the system is being constructed to handle. Many of these functions will span the entire spectrum of major program blocks, wherein, only certain parts of the functions are provided in each block. This is a laborious task, but crucial to the desired end result.

The task can most easily be accomplished while the software design is in-progress. The designer of each program block is the logical one to provide this function. This means that an additional task will be superimposed upon the program design, with a corresponding increase in manpower per block. It is assumed that the design is being accomplished according to the Structured Programming rules so that the overall objective can be met.

In addition, the gathering of information will have to be specified for the software designer, in terms of what information is to be collected during the design stage. He should be aware of how the logic he is developing is to be used so that he can make the appropriate correlations. A partial list of what might be submitted to him is shown below.

<u>Extended Call Features</u>

(1) Direct Access Dialing

(2) Abbreviated Dialing

(3) Call Forwarding

(4) Call Transfer

(5) Line Grouping

(6) Call Placement Restrictions

(7) Busy Diversion

(8) Camp-on Busy

(9) Group Hunting

(10) Remote Answer

(11) Attendant Recall

<u>Digit Reception</u>

(1) Dial Pulse

(2) DTMF

(3) MF

(4) MF Confirmation

Using the supplied list, the designer would then consider which modules in his design participated in any of the list items. The previous example of the scanning logic can be used to explain his duties.

The initial task of the software designer will be to identify the modules in his program block as was outlined in paragraph 4.1.3.3.3. Having done this, he can now begin the correlation process.

He begins by analyzing each list item, determining whether any of the logic supported the particular item, and denoting those which do. An example of a chart which he might prepare illustrates the technique.

	System Function	Modules
(1)	Direct Access Dialing	None
(2)	Abbreviated Dialing	None
(3)	Call Forwarding	None
(4)	Call Transfer	None
(5)	Line Grouping	None
(6)	Call Placement Restrictions	None
(7)	Busy Diversion	None
(8)	Camp-on Busy	None
(9)	Group Hunting	None

(10)	Remote Answer	None
(11)	Attendant Recall	SCL-A01; SCL-B01; SCL-C01; SCL-D03; SCL-B03; SCL-C05; SCL-C06
(1)	Dial Pulsing	SCL-A01; SCL-B02; SCL-C03; SCL-C04; SCL-D01; SCL-B03; SCL-C05
(2)	DTMF	None
(3)	MF	None
(4)	MF Confirmation	None

It can be seen from the preceding list that the scanning logic modules do not contribute to the implementation of the first ten functions. This means that if any change is made to those functions, or if they are not included within the total software system, that no changes or accommodations need be made for the scanning logic.

The first item in the list which is affected then is the Attendant Recall feature. It can be seen that all modules within Terminal Type 1 are used, except for modules which convert ASCII to Binary and ASCII to BCD and the Verify Input module, and that all modules under the Interprogram Data Transfer block are used but none under Terminal Type 2 are required.

The second list item which uses scanning logic is for Dial Pulsing. For this function, all of the modules under Terminal Type 2 are needed, one of the modules under Interprogram Data Transfer is needed, but none of the logic included within Terminal Type 1 is necessary.

Using this approach, it can be seen how all modules can be isolated with respect to their participation in the multitude of system functions, but it can also be seen that there is much writing to be done by the software designer to provide this detailed information.

In an attempt to reduce this effort, and to minimize the number of manual errors which could occur, a modified version of the task is envisaged. Recalling that in the previous section, it was decided to denote the various sub-branches within the structure, this idea can now be used.

It will be a common occurrence that an entire branch will be needed to fulfill a functional requirement. There will also be many situations which will require only a subset of the entire program block. Therefore, for the latter situation, it is necessary to specify the branches which are needed within the block to fulfill the desired function. The branch designators which are created, allow this to be done readily. Therefore, the module correlation can be modified, using branch indicators, to look as follows:

System Function	Modules
Attendant Recall	SCL-A01; SCL-B01-C; SCL-B03-A,B
Dial Pulsing	SCL-A01; SCL-B02-A; SCL-B03-A

The rule which is established and used here is that if any branch indicator is specified in the notation, then all sub-modules which carry that branch indicator are to be included. In the example for the Attendant Recall function, it is necessary to specify only the "C" branch of the Terminal Type 1 block since no other modules were necessary. The Interprogram Data block was reduced in specification since all modules were used.

In the example for Dial Pulsing, it is necessary to specify only the topmost module of the Terminal Type 2 block, and one branch from the Interprogram Transfer block.

The reduction in module correlation writing will increase the efficiency of this effort and reduce the time required to perform this function.

There is a further refinement which can be made to simplify the association process. This is the concept of grouping. It is defined in the following manner.

When a software designer constructs a major portion of logic, there are certain associations which he knowingly makes about his design. Those associations are usually implied, and not explicitly stated. For instance, the designer knows that when the Terminal Type 1 branch is used, it will always require assistance from the Interprogram Data Transfer branch. Therefore, the Terminal Type 1 logic is never sufficient by itself. The same is true in the example for Terminal Type 2 branch.

This relationship will exist many times over in the software design. Therefore, when it does exist, it can be explicityly stated in some manner, for information handling purposes during the System Generation process. It might be stated as:

If, SCL-B01-C, then SCL-B03-A,B

and If, SCL-B02-A, then SCL-B03-A

The previous correlation of functions to module relationship would then look like:

System Function	Modules
Attendant Recall	SCL-A01; SCL-B01-C
Dial Pulsing	SCL-A01; SCL-B02-A

This grouping concept allowed the elimination of the requirement to specify the Interprogram Data Transfer branch altogether. It is expected that in a major system development, this concept will reduce appreciably the module relationship entries.

As an extension of correlation process, assume that some feature, "X", required all of the logic within the scanning logic program block. All that would have to be called out then would be:

Feature "X"	SCL-A01

This would indicate that all modules at lower levels should be included.

There is another topic which needs to be addressed in this section. It deals with the multiple allocation of modules in memory.

When the software structure is drawn according to the structured programming criteria, it was of no concern that multiple representations of the same module were constructed many times. It was of some concern that the representation reflected the logic which had to be performed rather than where it physically existed. An examination of a large system would then most likely reflect several modules which performed the same function. It is foreseen that, in some cases, multiple copies would be desired.

There must be provided, therefore, some means which instructs the System Generation process in the manner in which to handle this condition.

The previous example reflects this situation for the BINARY to BCD conversion. This is the same module used by Terminal Type 1 and 2 but specified by two module identifiers. If the software designer wants this logic repeated twice, then he need not instruct the System Generation process in any other manner. Since the generation process works with module identifiers only, there is no indication that they contain the same logic, and thus, it will be duplicated.

However, if the module needs only one inclusion, the software designer must so indicate this requirement. This can be accomplished through an equivalency statement as follows:

Equate SCL-D03-C to SCL-D04-A

The System Generation system would then determine if both of these modules were called out in the correlation process and insert the logic only once. Of course, re-entrant procedures would have to have been applied, if necessary. It is assumed that the designer has taken this into consideration during the implementation.

4.1.4 Overlay

The designs currently being undertaken for both Message and Circuit Switches have evolved over the past decade to the point where these systems are reaching very complex magnitudes. During this same period, equipment has matured appreciably. The techniques applied to the solutions of these system problems, particularly concerning software development, are capable of using resources which were not heretofore available.

One of these areas concerns the use of overlay programs. It is becoming feasible to consider that certain portions of the software need not be resident within the main program storage areas, i.e., magnetic core or solid-state memory. Some programs may be used only occasionally and therefore, retained on some external media, such as disc or drum storage devices. When this is done, space must be provided within main memory which will accommodate different overlay programs at different times.

Care must be taken when overlay solutions to memory utilization is considered. An increased amount of system overhead will occur. This will take additional memory space, as well as the design of system protocols to involve this capability. However, the chief problem which must be considered is response time. It would be possible to configure an overlay capability in such a manner that the potential system throughput is decreased. Or, the system may, under some conditions, fail to respond in sufficient time so that it doesn't meet the intended objectives at all.

The main reason for addressing the overlay possibilities is in regard to program configuration criteria. A means must be provided to identify programs which should be configured to be always resident within the main storage areas of memory as well as those which can reasonably be expected to accommodate overlay handling.

The System Generation process will then have to automatically handle both overlay and non-overlay programs. Some method must then be devised to indicate this requirement. Additionally, the system will have to physically configure the total software system to be consistent with the overlay scheme. This will amount to positioning the program modules on the input media (tape, drum, disc, etc.), so that proper identification and loading can occur.

One final point is to be made for overlay programs. The system measurement capabilities will have to take into account overlay programs which contribute to the system loading. The method which is advanced for this analysis, should also be used for these kinds of conditions. However, transfer delay parameters may have to be inserted into the calculations to predict the overall effect of using this technique.

4.1.5 Executive Control

The executive program is a special case of the application program system in both Message and Circuit Switch environments. Because it is somewhat unique, some special procedures are foreseen in its handling by the System Generation process. The extent of any special handling will depend upon the kind of function which it is designed to perform.

4.1.5.1 Functional Overview

Typically, executive programs are designed to handle the scheduling activities of all other program modules within the software structure. This involves using some pre-established criteria by which programs are called into operation. Many methods have been devised to provide this kind of information.

In addition, many executive programs are also tasked with other system functions. For instance, the Input/Output functions may be relegated to the executive. It then becomes the chief program module which interfaces the common control subsystem with the other elements of the switching system through the transfer of the "real world" data. The executive may also be tasked with the duty of performing system maintenance activities, including switchover processes. Periodic checks would then be made by the executive to verify the operational capability of most of the system. Special sequences would be called into operation when these checks revealed error symptoms.

4.1.5.2 Special Problems

The make-up of the executive is not particularly important in the physical construction of the software system, except that portion which addresses scheduling. This area deserves further attention.

It will be possible for the executive to conform to the rules of module segmentation as is done for other software blocks. Individual program modules can be identified and correlated into branches as with other programs. The problem is not with this process, but in specifying the major program blocks which the executive uses, as well as indicating the preferred execution sequence that is to be followed.

To make this point somewhat clearer, consider the program structure which was shown in paragraph 4.1.3.3. Five application program blocks were depicted, each of which was independently associated with the executive program block. It was possible to consider each program block as a separate entity, parse its logic into a multitude of small modules, and interrelate those modules according to their logical branches. But there was no consideration given to how that program block interfaced with the executive, or, how the System Generation process could be informed of this relationship. This effort was deferred until the executive control was discussed.

4.1.5.3 Viable Solutions

It is apparent that some means must be employed to configure the executive with the other portions of the system. The task may be considered as fulfilling two objectives:

(1) Parameterize the executive for scheduling programs at predetermined times or conditions.

(2) Provide information to the loading process so that processor capability can be ascertained.

Since it cannot be precisely determined how an executive will be constructed in a given environment, it will be possible to explore only certain conventions by which the objectives can be met.

Assume that the scheduling process involves a table look-up process for determining which program block to schedule. This table then is composed of fixed entries which must be preset to reflect the scheduling patterns the system must accommodate. It might also occur that execution frequency information is contained in such a table. A representation of this table is shown below, using the modules which were previously mentioned.

PROGRAM BLOCK	FREQUENCY
Register Logic	12.5
Scan Logic	25.0
Matrix Logic	I
Translation Logic	25.0
Non-Register Logic	200.0

The table reflects each program which the executive is supposed to schedule. Corresponding to each program is frequency information which reflects the desired execution cycle. It is assumed that a system clock is implemented in some manner and used by the executive for timing purposes.

In the above example, the executive would schedule the register logic each 12.5 ms, the scan and translation logic each 25 ms, and the non-register logic every 200 ms. The "I", corresponding to the matrix control logic would indicate that this program is directed into execution by a system interrupt, rather than any timing criteria.

Some rules may also be applied by which the executive is to schedule these programs. For instance, it could be stated that the order is important, and therefore, the programs should be scheduled from top to bottom according to their ranked position within the table.

This would mean that at any period of time, when more than one program was supposed to execute, the program which was highest in the table would be called first.

This rule might also apply to programs scheduled by interrupt, such as the matrix control logic. Since it occupies the third position in the table, it would be activated only if lower programs specified in the table were operating. That is, if any program order higher was operating, the interrupt would not be immediately honored.

It would be possible to inform the System Generation process of this scheduling arrangement by instructions such as those shown below.

```
Scheduled Order = RGL - A01, 12.5; SCL-A02, 25.0;
                  MXL - A03, I; TRL - A04, 25.0;
                  NRL - A05, 200.0
```

The entries would then be processed and placed into the table for executive program scheduling use.

The table structure used for scheduling can be extended further to include additional information. Two more items might be desirable. One of these could be the interrupt level assignment which the application program is to use. This would occur if multiple interrupts were handled by the system. The level and their table ranking would then have to be specified.

Another condition could prevail which requires specification. Consider that a particular program block is required to execute only when certain conditions occur in the system. This block is, therefore, demand dependent. No association with interrupts is made, nor can timing criteria be made available for scheduling this logic. However, the program block must still be ranked as are all other program blocks. The scheduling table might then appear as follows:

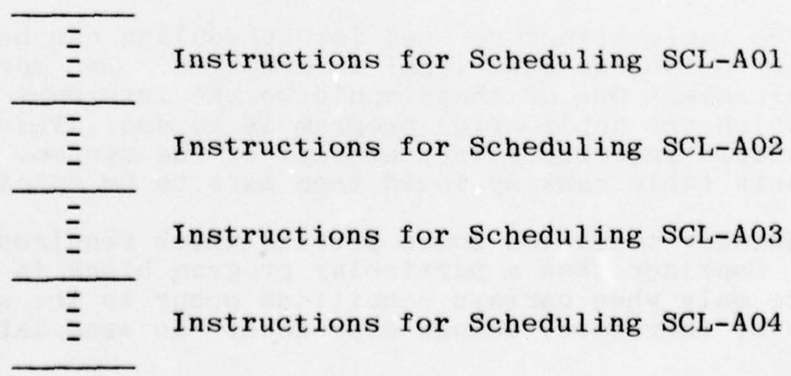
PROGRAM BLOCK	FREQUENCY
Register Logic	12.5
Scan Logic	25.0
Matrix Logic	I3
Translation Logic	D
Non-Register Logic	200.0

The table reflects the level of interrupt which is to be used for directing the matrix logic into operation. It also indicates that the translation logic is scheduled on a demand basis. The top to bottom ranking criteria would still be used in determining program execution order. An expansion of the scheduling order instructions given to the System Generation process could be developed to reflect these additional inputs.

There are, of course, several other methods by which an executive system can be constructed. One such method is In-Line code scheduling. This technique provides that the placement of the instructions used in the executive cause the scheduling sequence to occur. That is, the logic which enables the scan logic to operate proceeds the logic which drives the register logic, etc. The calling sequences are then different for each program block within the system. For this technique, the individual program segments would have to be uniquely identified and physically ordered so that they could be arranged in the correct manner.

A control structure for this type of scheduling implementation would look as:

<u>Executive Control</u>


Instructions for Scheduling SCL-A01
Instructions for Scheduling SCL-A02
Instructions for Scheduling SCL-A03
Instructions for Scheduling SCL-A04

Instructions for Scheduling SCL-A05

This structure provides the method for scheduling time and demand dependent programs. As the executive begins each new time cycle, the instructions it first executes cause it to schedule the SCL-A01 program block, followed by SCL-A02, etc. Logic within each scheduling segment must address the unique time requirements for that program block or demand conditions if the program is so driven. A method must also be constructed which provides interrupt control.

While this method of scheduling is perhaps some more efficient timewise in providing transfer to the appropriate program blocks, it suffers from being more rigid and inflexible to changes in the scheduling order. The System Generation user would have to be more cognizant of the physical implementation of the executive than with the first method outlined.

4.1.5.4 Correlation

The association of system functions to program modules which is performed for all program blocks may take another form when the executive logic is specified. This will depend upon the content of the executive; how it is put together and for what functions it is responsible.

If the system functions are provided entirely by the application program blocks, then the specification of required branches needs to be associated by some other criteria. This association was already provided if the first method of executive structure described in the preceding section is considered.

When the program execution schedule was prepared for the System Generation process, each program block which was to be scheduled was explicitly stated. The System Generation process may be made to use this data in some manner. This would be practical if the entire executive is to be used, since no branches were called out in that procedure.

For the situations which require the use of only portions of the executive, a relationship will have to be specified. This could be accomplished as shown below.

EXL-A01-A = SCL-A01

EXL-A01-B = NRL-A01

EXL-A01-C = RGL-A01

EXL-A01-D = TRL-A01

EXL-A01-E = MXL-A01

The above assignments indicate various branches of an executive system which correspond to particular program blocks. This would reflect program correlation rather than any system functional relationship. It would, nevertheless, allow for the inclusion of only those portions of an executive which are required for the system being constructed.

4.1.6 Overload Conditions

An important consideration which ought to be made for any system configuration is that of processor loading. A concept of System Generation has been outlined which allows for the inclusion of pre-programmed modules into a total software system. But what kind of loading will the processor experience when the system is in use?

Usually, when processor loading is considered, the main concern is with how much work the processor can undertake before it reaches execution saturation. In exploring this area, two conditions are of primary concern. First, it is necessary to know that the processor is not being overloaded, and second, how much loading does occur at expected peak levels.

Before exploring how it might be possible to predict loading data, the reasons why this is considered to be a particularly important parameter when a System Generation facility is developed should be explored.

4.1.6.1 Historical Trends

Historically, systems were created to fill some particular switching need. A detailed specification was provided which identified each function that was required during implementation. Using the specification requirements, the system designers then began to construct a system which fulfilled those needs.

Aside from the switching subsystems, (matrix, matrix control, line/trunk terminations, supervision detection, special consoles and instruments), the common control area was analyzed for the task in question. Of major concern here was:

(1) Which processor could best fill the needs?

(2) How many processors would be required?

(3) What kinds of common control interfacing should be utilized?

(4) How much memory is needed?

(5) In what manner should the memory be partitioned?

(6) What interrupt levels would be required?

(7) In multi-processor configurations:

  (a) What tasks should be assigned each processor?

  (b) How can data be exchanged between processors?

  (c) What plan should be followed for error conditions?

(8) How should the processor software be arranged?

(9) What maintenance/diagnostic capabilities need to be provided?

There were, of course, many other considerations which were taken into account during the design of each system.

One of the key areas which was addressed, but which was most difficult to definitize, was loading of the common control subsystem, or, overloading to be exact. There was certainly an attempt to predict these data, particularly when the number of required processors was addressed. These data were usually based upon some previous system experience, which to some degree approximated the requirements of the current system under design. When no previous system closely resembled the current undertaking, these data were extremely difficult to predict. The use of this process has seen some very accurate predictions, as well as some monumental catastrophies.

4.1.6.2 Loading Criteria

The loading of a processor is primarily dependent upon the external demands being placed upon it. These external demands cause some process to occur, depending upon the task involved. In a Message Switch Application, incoming message arrival rates and delivery requirements are the prime external demands. In a Circuit Switch Application, call placement rates and disconnect requests constitute the external demands. These external demands can usually be predicted rather accurately, when the environment in which the switch will operate is known. In many cases, the specification to which the system is being designed contains this information. The "front-end" loading can, therefore, be ascertained with substantial accuracy.

But how is it possible to predict the amount of time which the processor's software will consume in handling these requests? This is the information which is of real interest, and that which is most elusive.

There are several contributing conditions which lead to the consumption of processor time. A few of these are named:

(1) Overhead - that portion of the software which is required at all times, even during zero traffic conditions.

(2) Application Program Size - the amount of logic which needs to be executed to provide call/message handling- this will vary according to call/message type and any special handling required.

(3) Code Execution Repetition - the sections of software (usually loops) which are repeatedly executed, although they may be quite small.

(4) Administrative Functions - those non-message/call handling functions which must be performed immediately, but which do not provide service for external demands.

(5) Maintenance Functions - that logic which must be performed when fault indications are received. This is aside from the normal system checking which is periodically performed.

(6) Scheduling - that portion of the system which decides the order of program execution, and which itself contributes to the loading.

Typically, the loading is determined "after the fact"; that is, following the completion of the software design. A number of techniques have been applied at this point, most of which have had some success in determining these data. Hopefully, the system is not overloaded, but is loaded sufficiently so that the system was not grossly over-equipped. The latter is usually not the case.

4.1.6.3 Prediction Techniques

Returning to the topic in question now, it would be possible to continue the same process for loading predictions as was discussed above. But this process seems very restrictive and inadequate for the type of System Generation being pursued.

It is assumed that a System Generation facility would be based upon evolutionary developments. Specifically, software which is initially developed forms only a base for future additions, and that as each addition is created, it will be included into a resevoir of software modules. A subset of the total software capability could then be drawn upon to satisfy some particular switching requirement.

This concept precludes the possibility of drastically changing the system requirements. Otherwise, a completely new facility would have to be constructed, invalidating that which was already done. The systems must then be generic, so that a gradual evolution can be maintained.

Assuming that this commonality can be achieved, consideration must be given to what methods can be constructed to predict processor loading. It will be possible to draw upon the developed software modules and to form a viable software package. This means that hundreds of combinations of modules could be configured and installed for some switching configuration. It would be convenient to be able to specify, through the System Generation facility, the exact requirements for a particular switch, provide for all of the associated modules to be arranged together according to some predetermined scheme, and have the loading data automatically computed in some manner.

This seems to be a difficult goal. But one which can, perhaps, be achieved nevertheless. Remembering that the software was constructed of small modules, and that the ability to configure parts of the system together is provided, all that remains is to associate some execution time per module and to indicate the sequence which is to be followed. It then seems possible to build into the System Generation facility, the capability to use this information for loading determination.

The first step then is to place another task upon the software designer. That task will be to determine, on a module basis, the time required for its execution. Since the modules are relatively small, this is no large task for each module, although the composite will be substantial. If the typical design process is observed, in many cases the software designer does this himself, and for his own benefit during the implementation. Much of the information is then available, although requests are not often made for it. This information should be requested at the onset of the design task.

The second step will be to define the execution sequence which should be employed. That is, how often does each particular module execute in a specified time period.

This information is more manageable if the branches are considered, rather than individual modules. It has already been established that the branches can be identified, each of which contains several modules. An effort should then be spent to determine branch timing. This is only a summation of unique module times. No doubt this effort could be performed automatically, since all module times and all branches were heretofore defined.

With some method then, it has been possible to accumulate times associated with every branch in the system. The frequency of branch execution will then be required to determine loading across the specified time period. It should be kept in mind that only a selected set of branches will be utilized in this process, specifically, those which form the software for the configuration required.

As was mentioned earlier, the prime consideration on processor loading is to determine whether the system is overloaded. This will undoubtedly occur at peak busy hours if at all, so that is the loading level in which there is the most interest. This maximum handling situation is what has to be specified in some manner. This topic is further addressed in paragraph 4.3.3.1, Simulation. The details of a method are advanced to solve this problem.

4.2 SYSTEM SPECIFICATION

The System Generation facility under consideration is intended to satisfy those requirements peculiar to Circuit Switch and Message Switch configurations. It is not apparent that the same framework could not be used for other types of applications as well. The key in determining the approach suitability might be based upon whether evolutionary software is required for other systems, rather than in the techniques to achieve this end. It is expected that modifications could be introduced to provide variances in requirements if they exist.

The scope of this discussion, however, is in the constructs of a facility for switching applications. The main concern here is with possible system requirements, with which to tailor the implementation. Certain requirements address the configuration hardware, while others address the functional endeavors. Both of these require handling by the System Generation facility.

4.2.1 Physical Configuration

The physical make-up of the system will have to be specified for each software system which is to be prepared by the System Generation process. In order to do this, a base of hardware elements which can be drawn upon has to be specified. From this base, a subset can then be drawn, which will specify an exact hardware configuration to be used for constructing an operational system.

The primary purpose in constructing the hardware configuration is to enable the System Generation process to verify that the proposed hardware meets all of the functional capabilities which it is to undertake. A correlation, therefore, will be performed. This correlation is considered in detail in paragraph 4.2.3. This information is used by the System Generation process when the required capabilities are being assembled.

The list below reflects the kinds of hardware elements which would be used during this process. It consists of elements which relate to switching hardware modules and those which are included within the common control subsystem.

Switching Dependent Elements

(1) Number of line terminations - will specify the maximum capacity which the system is expected to handle.

(2) Type of line terminations - will specify the kinds of line terminations which can exist in the system.

(3) Number of trunk terminations.

(4) Type of trunk terminations.

(5) Number of register terminations - will specify the maximum number of receiver/sender units which the system can handle.

(6) Type of registers - will specify the individual kinds of registers which can be used in the system.

(7) Conference bridges - will specify the type, size and number of bridges allowed in the system.

(8) Matrix interfaces - depending upon the matrix and associated control, this will specify the interface which is to be handled with that subsystem.

(9) Special terminations - will specify types and maximum numbers of terminations, such as:

  (a) Recording

  (b) Paging

  (c) Inter-matrix

  (d) Encryption Modules

  (e) Echo Suppressors

(10) Attendant positions - will specify the number and type of special position equipment.

(11) I/O channel characteristics - will specify unique qualities of the I/O subsystem itself.

Common Control Dependent Elements

(1) Communications Processor Units - the number of CPU's which can be configured within the system. This will include the maximum number of CPU members, if a compatible family of CPU's exists.

(2) CPU Arrangements - the physical configuration which will be applied when more than one CPU is required.

(3) Memory Configuration & Sizes - the plan which is to be used for memory assignment, and the amount of such memory.

(4) Peripheral Units - will specify for each type of unit the number, capacity, speed, etc., which can exist.

(a) Disc Units

(b) Tape Units

(c) Printer Units

(d) VDU Units

The lists given above are only a sample of that actually handled by the System Generation facility. It is expected that several dozen such entries would be made a part of the hardware base. It is further assumed that there would exist no upper limit for this base, and that, as the system evolved, items could be added or deleted as necessary.

4.2.2 Functional Configuration

There would be another list constructed and input to the System Generation facility. This list would include all system capabilities which are possible to be handled. This list is the same as was used in the correlation process previously mentioned. In that process, the software logic as it satisfied unique system requirements was pulled together. This list will also be used to correlate those same functional requirements to the physical configuration of the system. All aspects of the total system, both hardware and software, will therefore have been considered.

This section serves to identify certain functional capabilities which are expected to be implemented with a Circuit or Message Switch environment. Again, this is not a comprehensive list, but one which could be used as a base on which further capabilities could be added.

Functional Capabilities

(1) Numbering Plan - the address dialing formats which may be called upon for use. This would normally identify seven and ten digit plans which specify area and office address codes and directory numbers.

(2) Trunk Group Arrangements - the manner in which trunk terminations are grouped for outgoing route selection, and for incoming traffic handling. Maximum numbers per group would be given here, as well as allowable sizes within a group.

(3) Traffic Collection Process - the activity which accumulates various call/message statistical data

during on-line operation. This category would include many subsets of individual system capabilities, each of which could be independently specified.

(4) Precedence Level Handling - the assignment of priorities by which calls/messages are to be handled. The number of levels possible and some criteria by which they are to be processed would be given.

(5) Conferencing Types - the various types of conferencing available within the system would be called out. This would include pre-set, broadcast, progressive and meet-me and any variations allowable.

(6) Signaling Requirements - this would specify every type of system signaling which is possible such as dial pulse, DTMF, MF, MF Confirmation, etc.

(7) Encrypted Call Types - this class would call out unique characteristics for logic which handled secure call/message placement.

(8) Extended Call Features - this section would specify all unique variations of call placement, including:

   (a) Direct Access
   (b) Abbreviated Dialing
   (c) Call Forwarding
   (d) Call Transfer
   (e) Line Grouping
   (f) Call Placement Restrictions
   (g) Busy Diversion
   (h) Camp-on Busy
   (i) Group Hunting
   (j) Remote Answer
   (k) Attendant Recall

(9) Message Length - the number of characters which can be expected to be sent for handling within one stream.

(10) Message Block Structure - the structure of messages themselves, in terms of field identifications, content type, and relative positioning.

(11) Routing Indicators - the characters included within a message which are used for destination route selection.

(12) Character Format Accommodation - the number and types of formats which can be processed by the system.

(13) Code Conversion Schemes - the requirements which address the transformation of character code from one representation into another.

(14) Format Validation - the requirements for verifying the character placement and content type prior to its utilization.

(15) Header Parsing - the activity which is required to break the message data content from preceding information used for process handling.

4.2.3 Correlation

The System Generation facility will be built to use the physical configuration and functional configuration lists in such a way to verify that the requirements match the actual capability. What is envisaged, therefore, is a process which ties certain hardware elements or subsystems to particular functional requirements.

It is expected that the correlation process would be a manual effort. Each functional capability would have to be compared with hardware elements within the physical configuration list to determine related hardware necessary. This process is similar to that which was performed in associating software logic to functional requirements. In the hardware correlation, however, there may not always be a matching hardware element. For instance, the capability to introduce abbreviated dialing is a software oriented implementation, and does not require supportive hardware elements. Neither does the capability to provide call forwarding privileges to certain subscribers. But the requirement to provide conferencing, of any type, has associated with it both software logic and hardware elements. The latter is the kind of correlation which is of concern at this time.

A correlation would then be made to indicate system functional dependency upon hardware elements. An example is shown below.

	Function		Hardware
(1)	Numbering Plan - 7 digit	=	Line or Trunk Terminations
(2)	Trunk Group Arrangements	=	Trunk Terminations
(3)	Traffic Collection	=	Tape Unit
(4)	Precedence Handling	=	None
(5)	Conferencing	=	Conference Bridge

This kind of information would then become part of the System Generation facility, which could be modified as system features or equipment needs change.

4.3 USER INTERFACE

Some consideration must be given towards the use of the System Generation facility. Many of the tasks which will be performed are built into this process and thus, are automatic. However, it is still envisaged that the entire process should be guided by personnel knowledgeable in the functions being performed. It is imperative then that this interface is oriented towards projected user needs in order to make an efficient and reliable operation.

There are two areas of concern, therefore, equipment used by operations personnel and procedures which they will follow in controlling the process. These topics are considered in the following sections.

4.3.1 Equipment

The hardware which will be used for the System Generation facility does not appear to require unusual configurations, nor to consist of elements which are themselves unique. The task which is to be accomplished with this facility is primarily data processing oriented. Files from mass storage devices will be accessed, merged with other files, and output on some suitable media. The internal processing which occurs is expected to be that of accepting user inputs and accessing files, where correlation functions are performed, with periodic results returned to the operations personnel.

Therefore, a system is envisaged which contains a central processing unit, magnetic tape drives, disc units, a line printer, and some equipment which allows user interface with the ongoing process. It is not apparent that a "batch" type processing operation could be utilized efficiently.

The user interface is perhaps the most important element with this configuration. It would provide the use with interactive capability to initiate the process, and to control and monitor that process while it was being performed. A CRT terminal device is probably the most efficient from a user viewpoint. In the interactive mode, intermediate results could be displayed for inspection. Decisions which have to be made during the process could be easily implemented at such a terminal. Although a hard-copy output is not considered to be essential during the process itself, it would nevertheless aid in providing a reference during the operation.

4.3.2 The Process

It is now possible to explore the user procedures which could be employed in fulfilling the final goal of the System Generation facility, the overall process by which this is achieved, the functions performed at each step, and the manual intervention deemed necessary.

Prior to exploring the operations by which the user controls the system, it is necessary to gain a clearer understanding into the process itself. Since the end result of the System Generation process is a "tailor made" software system, a multitude of software modules must be pulled together in some logical fashion. This function cannot be done in a vacuum. In order to do this, other elements must be defined, specifically, which hardware modules will be needed to fulfill the stated switching role and the system functions and features which the system is required to handle. These inputs will form the basis by which specific software modules will be selected.

This process is presented pictorially in Figure VI-3-1.

The processes shown in Figure VI-3-1 will be discussed before elaborating in detail about each step.

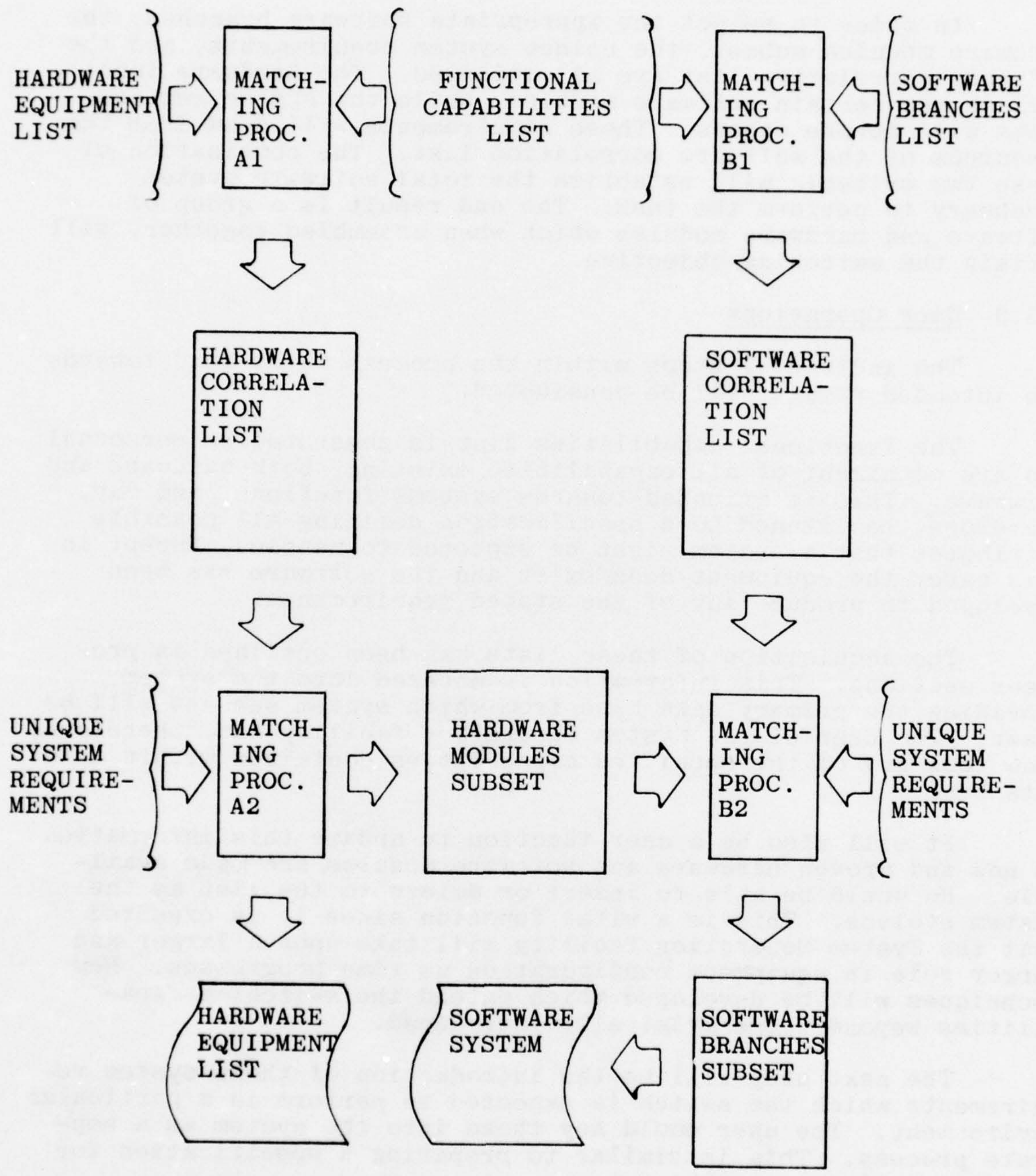
The key item in the initial process involves the creation of two correlation lists, one for hardware, and the other for software. To arrive at these lists, all hardware and software modules are matched with known system functional capabilities. This process (A1 and B1) yields an association of modules to capabilities. For instance, a functional capability to outpulse MF digits would be reflected in both the hardware and software lists. For hardware, this would associate MF digits outpulsing to MF sender units. The MF sender unit may also require a particular interface to the processing subsystem. This hardware module would, therefore, also be included. The software requirements for MF sending would likewise be reflected in terms of one or more software branches necessary to perform that function.

It is not of concern at this time, how many modules, either hardware or software, are necessary. Only that the correlation reflects all that are necessary for each individual capability.

The capture of this information, allows progression to the next process step.

At this point, the major concern is with the grouping of hardware modules to produce the physical equipment necessary for a particular switch configuration. The unique system requirements are introduced here. The matching process, A2, combines the hardware correlation list with those unique requirements to specify the actual hardware to be used. This process will yield a hardware equipment list matching the unique system requirements specified.

A second output of the A2 process is used as additional information in the selection of specific software branches to



USER PROCEDURES
SYSTEM GENERATION TECHNIQUES

FIGURE VI-3-1

fulfill the system requirement.

In order to select the appropriate software branches, the hardware modules subset, the unique system requirements, and the software correlation list are all utilized. The hardware input will invoke certain software modules, while the system requirements will invoke others. These requirements will draw from the resources of the software correlation list. The combination of these two criteria will establish the total software system necessary to perform the task. The end result is a group of software and hardware modules which when assembled together, will satisfy the switching objective.

4.3.3 User Operations

The individual steps within the process which lead towards the intended result will be considered.

The functional capabilities list is generated by personnel who are cognizant of all capabilities existing, both hardware and software. This is oriented towards systems functions, and may, therefore, be likened to a specification defining all possible attributes that a system might be expected to handle. Except in this case, the equipment does exist and the software has been developed to produce any of the stated requirements.

The acquisition of these lists has been outlined in previous sections. This information is entered into the system, providing the primary data base from which system subsets will be drawn. The user of the System Generation facility can, therefore, draw upon any of the specified capabilities contained within this data base.

It will also be a user function to update this information as new and proven hardware and software modules are made available. He would be able to insert or delete to the list as the system evolves. This is a vital function since it is expected that the System Generation facility will take upon a larger and larger role in equipment configuration as time progresses. New techniques will be developed which extend the switching capabilities beyond those originally introduced.

The next step will be the introduction of those system requirements which the switch is expected to perform in a particular environment. The user would key these into the system as a separate process. This is similar to preparing a specification for off-the-shelf procurement.

There are many approaches which could be considered here. One is simply a random input of capabilities which the system is expected to handle. It would not matter which item was first introduced since the system would be expected to validate the inputs and choose the appropriate equipment after all entries were made. However, the random input lacks any semblance of order, and thus, would most likely require further information or corrections in subsequent entry procedures.

It seems more plausible that some order of entry should be considered. One feasible approach would take the form of a question and answer format, in which the system queried the user in successive steps. If this was done, the entries would be considered sectionalized. For instance, all entries about the end user equipment could be introduced first, followed by supervision procedures, special consoles, etc. Using this approach, the input process would be simplified by progressively displaying all possible end user equipment available, all supervision procedures, all special consoles, etc. The user would select those which matched his specific requirements and would enter the number of each that was needed. The physical arrangement of the equipment could likewise be specified if necessary.

At some point in this process, the user begins to specify the equipment which is commonly shared and traffic dependent. Two possibilities exist here. First, it might be initially impractical to automate traffic calculations into the System Generation facility. The user would, therefore, use some predetermined knowledge in specifying the numbers of these units. The system would be equipped with whatever the user specified.

However, after some period of time, it might be deemed more desirable to include within the System Generation facility some traffic calculations which provided aid to the user in specifying numbers for traffic dependent equipment modules. Based upon the expected traffic, the system would call out for a calculated number of modules. The user would be able to modify these numbers based upon some additional information.

For the above process, the matching process, A2, would continue for the hardware throughout all of the user/system interaction. At the completion of the hardware related phase, a complete hardware equipment list would be printed.

Although this procedure outlined for hardware specification need not necessarily be made a part of the System Generation facility, some manual process would otherwise have to be substituted. This is possible, but the penalty is that the specification of the hardware and software would be done under separate processes, with the possibility of inconsistencies being introduced.

It would be valid, of course, to eliminate the hardware specification procedure in the event that no changes to the equipment configuration were necessary. This situation would exist if the same hardware was applied in a different manner through software reconfiguration only. The System Generation concept should not preclude this possibility.

The next process which the user would be undertaking is the specification of software capabilities. Those functions which tie the entire system together, to provide the end result, need to be introduced. This process parallels that which was undertaken for the hardware modules. In fact, many of the system functions, which required modules of hardware, will require counterparts of software.

Some elaboration is needed at this point. It can be said that the mere introduction of unique system requirements is not sufficient for the specification of software. The input from the hardware specification will describe the actual equipment used. These inputs are used to invoke appropriate software modules such as particular supervision procedures, matrix interface procedures, and particular peripheral equipment software interface logic. These are necessary so that the correct equipment interface procedures are included in the end product.

The specification impact from the System Generation user for unique system functions does not address equipment characteristics, but rather those user functions which the system is being constructed to provide. Therefore, the kinds of inputs which are considered here are features oriented, such as abbreviated dialing conference privileges, numbering plan, message characteristics and formats, and timing characteristics.

It is expected that there will be many system functions available from which to draw. It would seem logical to implement some order of entry which blended with other organizational procedures in which the System Generation facility is used. It should be, however, comprehensive and simplified to enable the user to specify easily the desired system.

Having entered all of the system functional capabilities, and effecting the acquisition of the appropriate software modules, the final user operation addresses the configuration of these modules. Some order already does exist by virtue of the manner in which the branches were originally constructed. This was covered in previous sections. What is important here is introducing execution cycles which the software is expected to maintain.

The executive program is the chief means by which scheduling will occur. A possible method for assigning execution cycles, introduced previously, effects this order assignment. The user would be interested in ascertaining whether the configuration he has produced is effective in meeting the traffic conditions which the system is expected to handle. It would be possible for instance, to select the appropriate software modules, but to arrange them in several different configurations. Some of these configurations may not be optimum in meeting the known requirements.

What would be advantageous is for the System Generation facility to verify in some manner the throughput capability. Several configurations could be simulated under the direction of the System Generation user and a selection made which best fits the requirements. This capability is addressed in the next section.

One additional task which the System Generation user would be concerned with is overlay software logic. It may be desirable for certain segments to be rolled-in at particular times rather than being permanently resident in main memory. A knowledge of the software system organization would need to be known in some detail to allow a workable system arranged with overlay logic.

The user would need to know, for instance, which software blocks are capable of being dynamically allocated to storage space, and the size of each block which could be so handled. Another important consideration in allowing or disallowing overlay procedures concerns what penalities, if any, would be incurred from this implementation.

Each block which provided overlay techniques to be applied would have to be tagged with some conditional information specifying criteria mentioned above. The System Generation user would then make a decision as to the advisability of each block in question.

4.3.3.1 Simulation

Many kinds of simulations exist which perform various roles in systems design and test. Regarding the System Generation facility, it is not the concern with attempts to verify correctness in logic or code. It is presumed that prior to the inclusion of any software branch into the software branches list that these types of verfiications would have been made. The insertion of faulty programs is, therefore, not anitcipated.

The primary concern is with the effectiveness of the system put together in this facility. Specifically, the effectiveness in terms of throughput. This must be equated to individual program execution times.

To accomplish this end, further information must be gained about the execution times of individual branches which constitute a program. This would be a composite of all execution times of the modules which composed a branch. The problem which surfaces here is two-fold. First, as noted previously, the programmer would need to calculate approximate times per module. This information would be collected and entered at the branch level. This requirement does not become particularly difficult when it is considered that individual modules are relatively small. Each module is performing only a small role in the total system task.

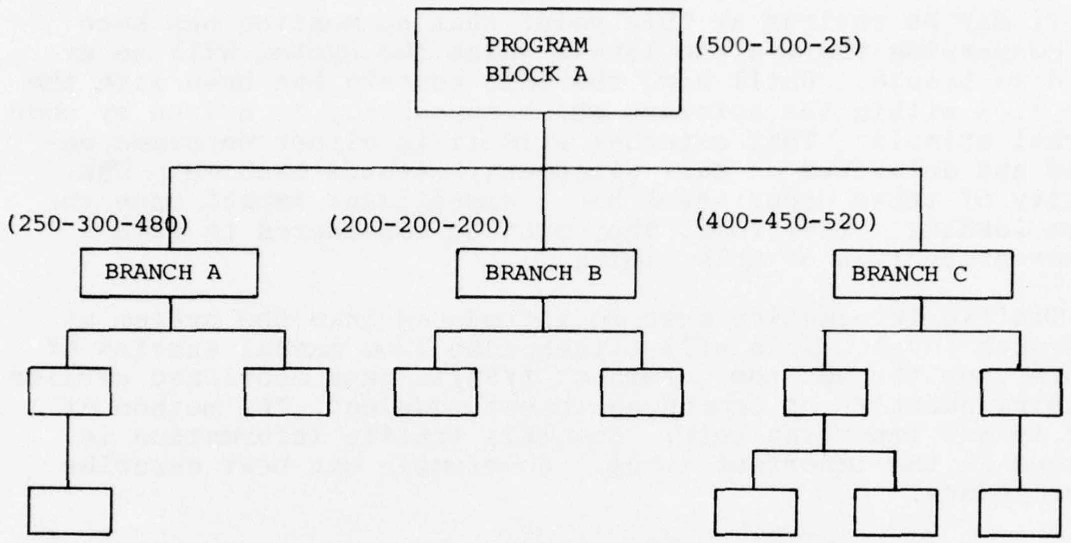
Following the module coding, it is not foreseen that much effort would need to be expended in determining the execution time.

The second problem is perhaps more difficult and time consuming. Consider that for any particular branch that several modules exist, and that only certain modules may need to execute when that branch is entered at a particular time. Some means must then be determined to capture the normal execution time per branch.

Since the main concern is with approximations, as stated before, this program can be simplified. It will take some effort, but the pay-off can be significant. Normally, each branch can be determined to have an execution sequence which causes minimal execution times and also maximum execution times. These can be determined by inspection and recorded. It is also conceivable that the path through a branch may also have a normal execution route, which is either one of the two extremes, or somewhat in between the two. These times would represent variances in handling the function which the branch was designed to perform. That information can then be captured.

Therefore, three distinct branch execution times can be accumulated; best case, normal case, and worst case. These times are then associated with each branch placed into the software branches list. The System Generation facility could then make use of this information in an automated way to calculate the associated execution times of a specified software configuration. This is possible since all branches are known, all things per branch are recorded, and the configuration is defined. A summation of these times across some defined interval will deliver approximate loading values. The calculations could be made for all three specified branch times, or as a mix of combinations of those times based on some probability of occurrence.

A structure which reflects the timing of three branches is shown below.



Considering Branch A, the least execution time taken by the logic is 250 us, the normal use is 300 us, and the maximum time is 480 us. Note that Branch B timing remains the same for all three cases.

The next step in determining processor loading concerns the frequency of execution of each branch of the system. It must be known how often a particular branch is entered in order to compute the total amount of processing time across some predefined interval of real time.

Fortunately, some of the information necessary to predict this has already been generated. Recall that in the discussion of the Executive Control, the program block frequency needed to be specified. This information can now be used by the simulation process to calculate execution frequency. This works well, except for program blocks which are either demand or interrupt driven. For these blocks, it is necessary to specify, for simulation purposes, the frequency with which they will operate. This will require a judgement by the system designer. The method advanced for branch timing, giving three values, could be made use of here as well. A knowledge of the system is important in order to ascertain the most realistic procedure to follow. This information is shown on the previous diagram. It indicates that Program Block A is executed normally every 100 milliseconds; that at least it is executed every 500 milliseconds and no more frequently than every 25 milliseconds.

This program block might represent one which is normally driven by demands or interrupts, but for simulation purposes, has been tagged with this execution frequency data.

It may be obvious at this point that no mention has been made concerning the traffic levels which the system will be expected to handle. Until now, the main concern has been with the logic flow within the software which supposedly is driven by some external stimuli. That external stimuli is either messages received and delivered or call placement/releases handled. The quantity of these occurrences has a significant impact upon the system loading. Therefcre, they must be considered to gain a clearer projection of this timing.

Traffic information must be introduced into the system at the branch level. This will either come from manual entries of the user, or through the automated traffic data mentioned earlier regarding quantity of common equipment modules. The method of input is not important here. How this traffic information is utilized is the important issue. An example can best describe the procedure.

Assume that the system being constructed is a circuit switch application for 1000 subscriber terminations. Assume further that the system also has assigned to it 300 trunk terminations. The effect of the demands placed upon the software needs to be calculated to form a true loading picture.

One method to achieve this is to isolate the areas which are directly affected by call placements. These will include the front-end scanning logic, register logic including address collection and sending, address translations, and special features logic employed in handling certain calls. Other areas will have little or no increased loading when traffic levels are increased or decreased.

The determination of affected areas is not a particularly difficult task. This should be accomplished by the system designers, who are most cognizant of the applicable areas. Having isolated these areas, it is possible to assign weighting parameters to them along with the conditions by which these parameters are adjusted. For instance, a branch or branches which are affected by seizure requests would be assigned weighting values dependent upon traffic. Idle conditions will force this parameter to zero. Half-load traffic, defined as "N" number of seizures per second would provide a second value. Full loading, "2N" would force the parameter to a higher value. Intermediate values could likewise be calculated.

The parameters are then used to modify the individual branch timings calculated before. Some educated judgement is necessary at this step to provide a realistic estimate of the parameter usage. It may not be realistic, for instance, to double the time of scanning when the system is fully loaded. It may only be necessary to apply a weighted value of, say 1.3. The rationale for this lies within the scanning design itself. It is conceivable that the majority of the scan logic occurs independent of traffic load, as was shown in Section 2, paragraph 2.4.3.2; and thus, only small additional amounts of processing are required to handle new seizures.

Another example would be in the access logic necessary to operate the matrix. In this case, the loading might be substantially increased by call placement activity. Assuming that each call requires "N" accesses as an average to the matrix, then each call will increase the loading by an equivalent amount. Again, this determination is design dependent.

The concept which has been advanced for simulation of software loading is an attempt to approximate that which would exist in the real environment. This approximation could be developed in great detail so that a very accurate picture is determined.

No matter what level is attained, the end result would be to give the System Generation user the opportunity to configure the system in different manners to predict throughput performance. Using several simulation runs, the one which indicated the best overall performance for known criteria could then be chosen. The system would then produce the necessary output for system loading and corresponding documentation.

4.4 CONCLUSIONS

The concepts presented for the development of a System Generation facility provide a technique which, if implemented, would allow for the re-use of software, in total or in part, for varying configurations and functional requirements of switching systems. No attempt has been made to estimate the effort needed to be expended in achieving this goal. This estimate must be based upon a known configuration of hardware and software in order to be meaningful, and upon the degree of implementation which would initially be undertaken. It is not suggested that the initial attempt at System Generation should encompass the entire spectrum of the generation process. Rather, a more realistic approach would seem in order; one in which various portions of the technique were applied, and built upon in successive stages. However, early planning must be made for evolution into the total system.

A concerted effort must then be made at the onset of a system design in order to achieve a flexible and workable System Generation process. Planning must be introduced at an early stage of system development. One cannot reasonably expect that a system, already developed and in use, could be accommodated in the manner outlined within this report. The possibility to include certain portions might be possible after the fact, but a total system would necessitate a substantial effort which most likely would negate any subsequent pay-off.

Crucial areas which need to be considered at an early stage are:

- System Configuration

  The elements which compose the hardware modules, their interconnectivity, and modularity need to be uniquely specified. A rigid system which does not afford flexibility in expansion or contraction, nor additional add-on equipment, would not seem to be a reasonable system for which to construct a System Generation facility. Therefore, the ideal system would allow adaptability to changing configurational requirements.

- Functional Capability

  It can reasonably be expected that the functions which the system will be expected to perform will vary extensively from one configuration to another. Multiplicity of carbon copy systems do not require specialized configuration procedures and may be reproduced easily by other means.

- Software Structure

  The generally accepted procedures of Structured Programming must be adhered to for consistency of design and modularity. The failure to apply standard practices will yield an unworkable or burdensome task for the user to manipulate.

- Identification and Correlation

  The procedures outlined in this report need to be addressed during the software implementation to gain the input material for the process. Although this information could be derived after the design, it can be expected that an appreciable amount of effort would be needed if this task is deferred. The availability of knowledgeable personnel could then pose a serious problem.

- System Generation Software

  The manner in which this software is created, and the extent to which it achieves the desired goal must be carefully planned. This activity can parallel the switching design task if the entire system is well planned in advance. Configurations of data which will be available, expected operational procedures, and output from the process, must be specified in sufficient detail to provide the intended result.

While the concepts and techniques presented within this report should provide a basis on which a System Generation facility can be constructed, they are considered to be only a basis for which further exploration of the topic may be made. The framework has been provided in which further investigations may be pursued.

PREFACE

This volume, volume VII consists of five appendices.

Appendix I	-	Contains a description of various classical processor architectures.
Appendix II	-	Contains an analysis of the advantages and disadvantages of utilizing Content Addressable Memories in Circuit and Message Switching applications.
Appendix III	-	Contains a Glossary of Terms and Acronyms used throughout the final report.
Appendix IV	-	Contains a Bibliography of articles and reference books which were used throughout the course of the study.
Appendix V	-	Contains a summary of Circuit Switch and Message Switch functional breakdowns and calculations.

APPENDIX I

CLASSICAL PROCESSOR ARCHITECTURES

In order to determine the Communication Processor System which is economically best suited to serve the previously described applications functions, the following plan was adopted:

(1) The applications functions are discussed as if they are to be implemented on a single sequential processor (uniprocessor approach). Although it soon becomes apparent that this approach is not the practical solution for the problem, it offers a means to normalize the evaluation of other processor system architectures. The approach also lends familiarity, logical simplicity, and provides a wealth of historical emperical data.

(2) The necessity of increasing throughput leads to a search for "Parallelism", i.e., by processing several functions concurrently, or by processing the same function for several events (calls, messages) concurrently.

(3) The application functions are analyzed to determine if an advantage may be gained by use of an associative (content addressable) memory, and if further (speed) advantage may be gained by associative processing.

(4) The results of "optimizing" each application function with respect to several processor unit architectures will then be weighted and merged to evolve the recommended processor system architecture (Volume IV of this report) and the resulting processor unit architectures.

In order to avoid confusion between the terms processor unit and processor system, the following is offered.

A group of processing elements are considered a processor unit if they execute machine level commands from a single control device in synchronism.

A group of processing elements are considered a processor system if they contain separate control units, and execute machine level commands (relatively) asynchronously. Note that a common "Master" control unit can issue commands to several processing elements each having its separate control unit. This constitutes a system by our terminology.

Fundamental Processor Unit Architectures

It is pertinent to describe several popular processor unit types and briefly describe the properties of a processing function which best exploits the advantages (or disadvantages) of each. "Advantages" and "disadvantages" are relative to the uniprocessor which has been adopted as a "normalizer".

Uniprocessor Unit Architecture

Figure AI-1 shows a block diagram of the classical "general purpose" machine. Only one stream of instructions is operating on one set of data at a time. The fundamental means of increasing throughput is to reduce execution and memory cycle times. The limitations are therefore fundamental.

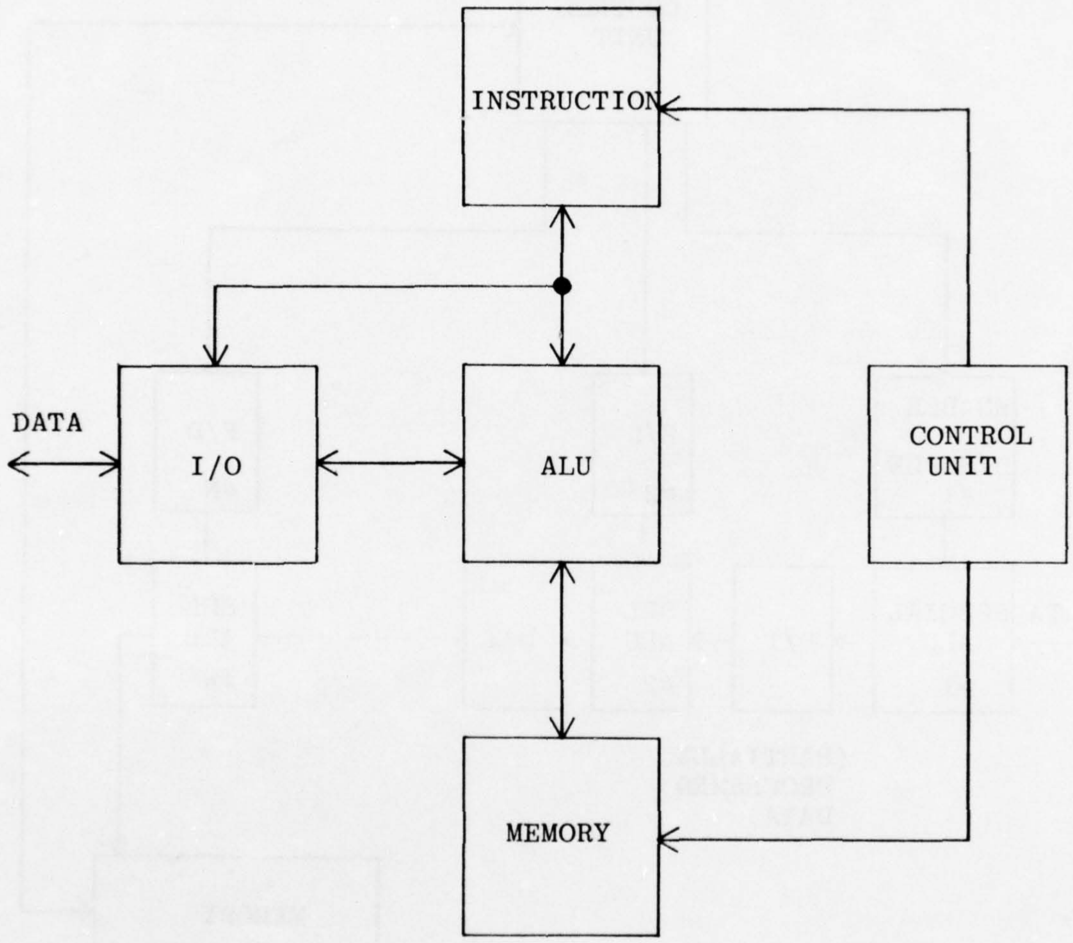
Two embellishments which have been utilized in the past to increase throughput are "Instruction Look-Ahead" and "Multiple Machine States or Register Sets".

"Instruction Look-Ahead" is a means of having the "next" instruction ready for execution immediately on the completion of the previous instruction, thereby completing two (or more) operations in one memory cycle time. Full exploitation of this concept requires an increase in the number of wires to the memory unit such that multiple instructions are accessed during one memory cycle time.

"Multiple Machine States or Register Sets" is a means of saving storage cycles in the case of transfer of machine control from one program to another. This construct can increase throughput if the nature of the process requires a significant amount of jumping from one (sub) program to another. In the case of fairly "long" routines within one program, the amount of time saved (on a long time average) by eliminating the storage cycle overhead is insignificant; in which case the only "advantage" offered by multiple registers is a possible programming convenience.

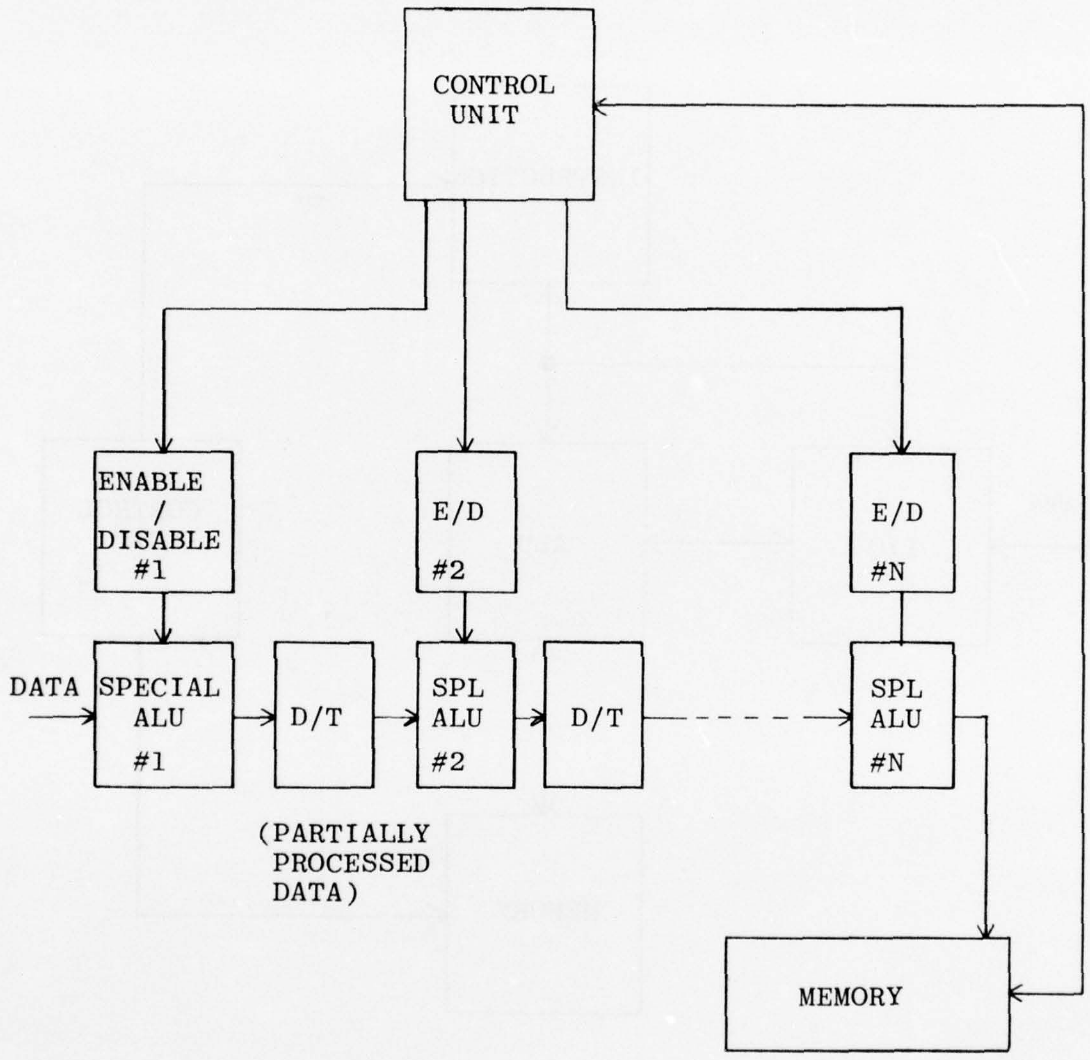
"ALU Pipelining" (Unit) Architecture

"ALU Pipelining", shown in Figure AI-2 is a means by which "complex" instructions may be mechanized to reduce exexution time. The method is especially useful in floating point arithmetic where a large amount of data must undergo the same processing. One special ALU (Arithmetic Logic Unit) is used solely to align exponents, another to multiply, another to add, etc. The instruction ADD from the control unit enables one set of ALU's and bypasses others, while the instruction MULTIPLY would enable a different set of ALU's as required. The advantage of this mechanism is lost if only logical manipulations are required. Logical operations are "simple" and can be



UNIPROCESSOR UNIT ARCHITECTURE

SINGLE INSTRUCTION STREAM - SINGLE DATA STREAM
(SISD)
FIGURE AI-1



"ALU PIPELINING" (UNIT) ARCHITECTURE

MULTIPLE INSTRUCTION STREAM - SINGLE DATA STREAM (MISD)

FIGURE AI-2

mechanized efficiently in one "stage" of ALU. "Complex" operations requiring several execution cycles can gain a speed advantage by this construct, but they must be executed often (statistically) if "ALU pipelining" is to be economicallv justifiable. Other benefits of an ALU pipeline organization may be gained if it is possible to establish multiple data paths through the ALU set with a minimum of bottlenecks (determined by simultaneous contention of one special ALU).

(Note: The term pipelining" was adopted from an analogy to the operation of a pipeline. The analogy and the term have also been applied to resource allocation in a multiple processor system. "ALU pipelining" is the only context in which we use the term.)

Parallel Array (Unit) Architecture

The Parrallel Array Unit Architecture, shown in Figure AI-3 is especially suited for applications where a large amount of data can be processed simultaneously. The processes must be (relatively) independent but similar. Each active Processing Element, PE, is synchronously performing the same operation on its particular data stream. Any PE can be activated or deactivated independently, but if it is active, it is performing the same operation as all the other PE's.

In order to exploit this type of processing, the data must be "synchronizable" to the process, i.e., the instruction stream sequence is data independent, and the data must be ready on demand. This implies that the data is not real-time, but is resident at the time of processing. (Note the difference to cases where the data is real-time, and the program sequence is the dependent variable.)

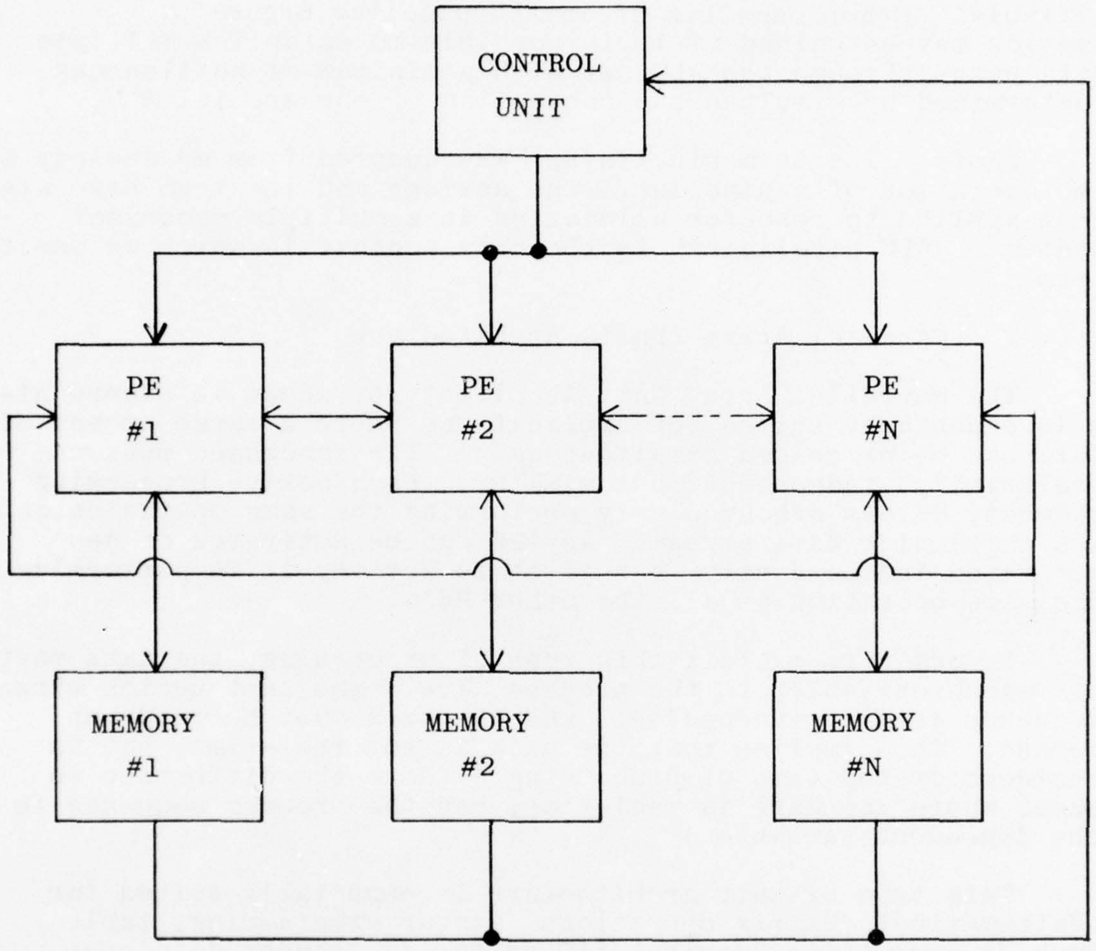
This type of unit architecture is especially suited for (Mathematical) Matrix operations, linear programming, table look-up operations, digital filter design, sensor data processing and pattern recognition.

Associative Array Unit Architecture

The Associative Array Unit Architecture, shown in Figure AI-4 provides a means of exploiting the properties of an Associative (or Content Addressable) Memory.

(1) Word Slice - An Associative Memory may be accessed by a word address, causing the data at that address to appear on the output lines. (This is the standard mode for conventional RAM memories).

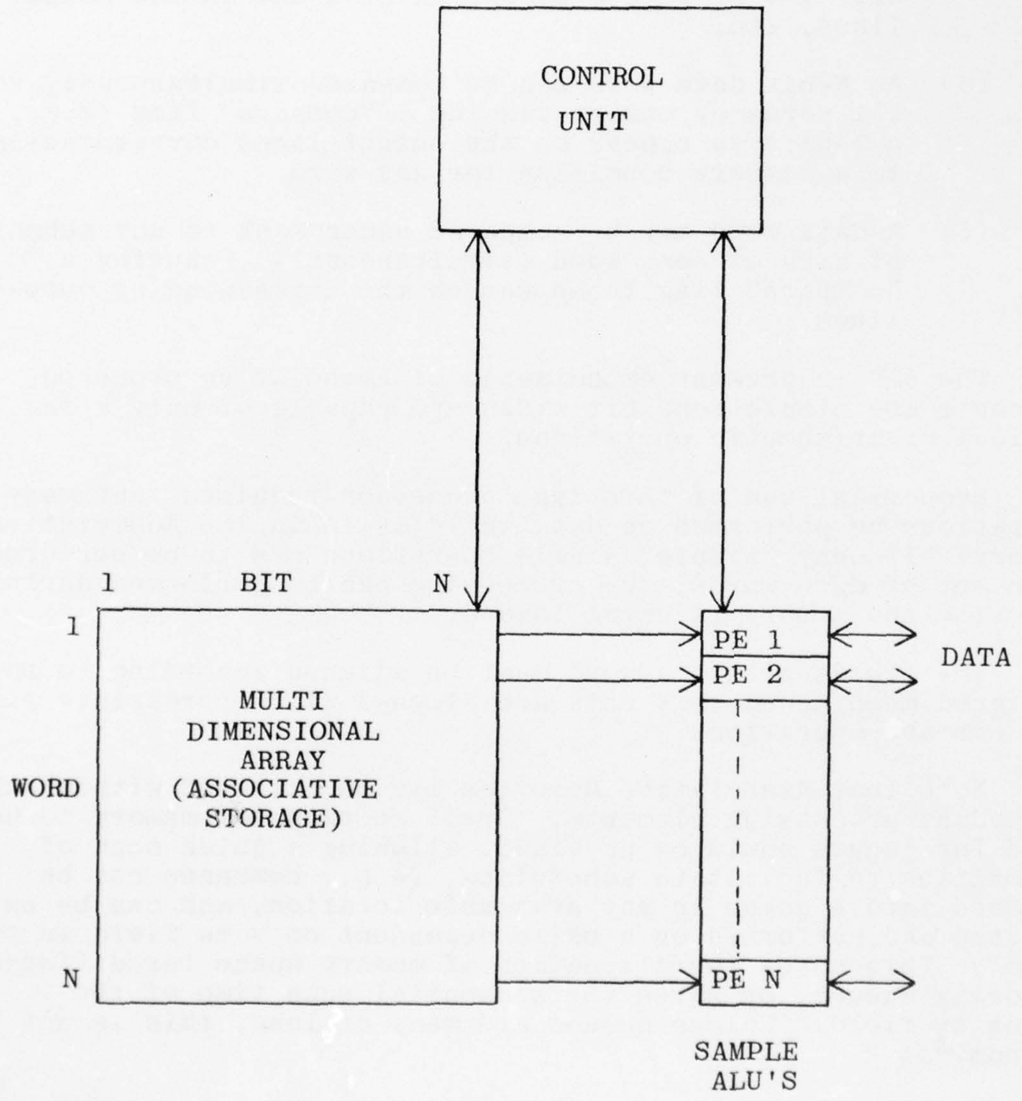
(2) Bit Slice - An Associative Memory may be accessed by a bit position address, causing the contents of each bit



PARALLEL ARRAY (UNIT) ARCHITECTURE

SINGLE INSTRUCTION MULTIPLE DATA STREAM (SIMD)

FIGURE AI-3



ASSOCIATIVE ARRAY UNIT ARCHITECTURE

SINGLE INSTRUCTION STREAM - MULTIPLE DATA STREAM (SIMD)

FIGURE AI-4

AD-A037 159 NORTH ELECTRIC CO GALION OHIO F/G 17/2
COMMUNICATIONS PROCESSOR SYSTEM.(U)
JAN 77 K HAGSTROM, B BEIZER F30602-73-C-0314
UNCLASSIFIED RADC-TR-76-394 NL

2 OF 3
AD
A037159

of various words to appear on its appropriate output line.

(3) Combinations of (1) and (2) are possible, e.g., Bits 1-4 of Word 1 appear as Bits 1-4 on the output lines, Bits 1-4 of Word 2 appear as Bits 5-8 on the output lines, etc.

(4) An N-bit data word can be compared simultaneously to all words of memory causing a "compare" flag (e.g., a 1-bit) to appear on the output lines corresponding to a compare condition for any word.

(5) A data word may be compared under mask to any subset of bits of each word (simultaneously), causing a "compare" flag to appear on the corresponding output lines.

The ALU in present embodiments of Associative processor concepts are simple (one bit wide) and capable of only a few logical or arithmetic operations.

Economical use of this type processor requires that many operations be performed on data while it is in the Associative Memory. If only "simple" single operations are to be performed on a set of data words, the processing can be performed during the time the memory is being loaded.

The fields within a word must be aligned according to some assigned meaning so that data are aligned with appropriate masks for compare operations.

Note that Associative Memories may be utilized without the attendant processing elements. Small sections of memory to be used for queues could be provided, allowing a quick scan of priorities to facilitate scheduling, (e.g., commands can be entered into a queue in any available location, and can be extracted and performed on a basis dependent on some field in the word.) This saves the allocation of memory space for different priority queues, or saves the sequential scan time of the priority field. Unless queues are many or long, this is not economical.

To date, Random Access Associative Memories have been relatively expensive to produce, and therefore, have been confined to small memories (e.g., 256 x 256 bits).

Tasks particularly suited for Associative Processing include Fast Fourier Transform operations, String searches, Matrix operations, EW pulse analysis, Track-while-scan (radar) and information retrieval.

FUNDAMENTAL PROCESSOR SYSTEM ARCHITECTURES

Two different Processor System Architectures are described here. It should be noted that combinations of these types are also possible.

Distributed Processor Array (System)

The Distributed Processor Array system architecture, shown in Figure AI-5, is characterized by the relatively independent operation of several processor units. Note that the Processing Units may be dissimilar types (e.g., Associative, Parallel Array, etc.) each contains its own program and data memory, and communicates to others by common memory buffers and queues or by interconnecting data links. A discipline must be established to prevent clashes when several processors contend for simultaneous access to the same memory area.

This type of organization is best suited for applications where processes are relatively independent (minimum amount of parameter passing between units) and when the processing load can be distributed to utilize each processing unit uniformly.

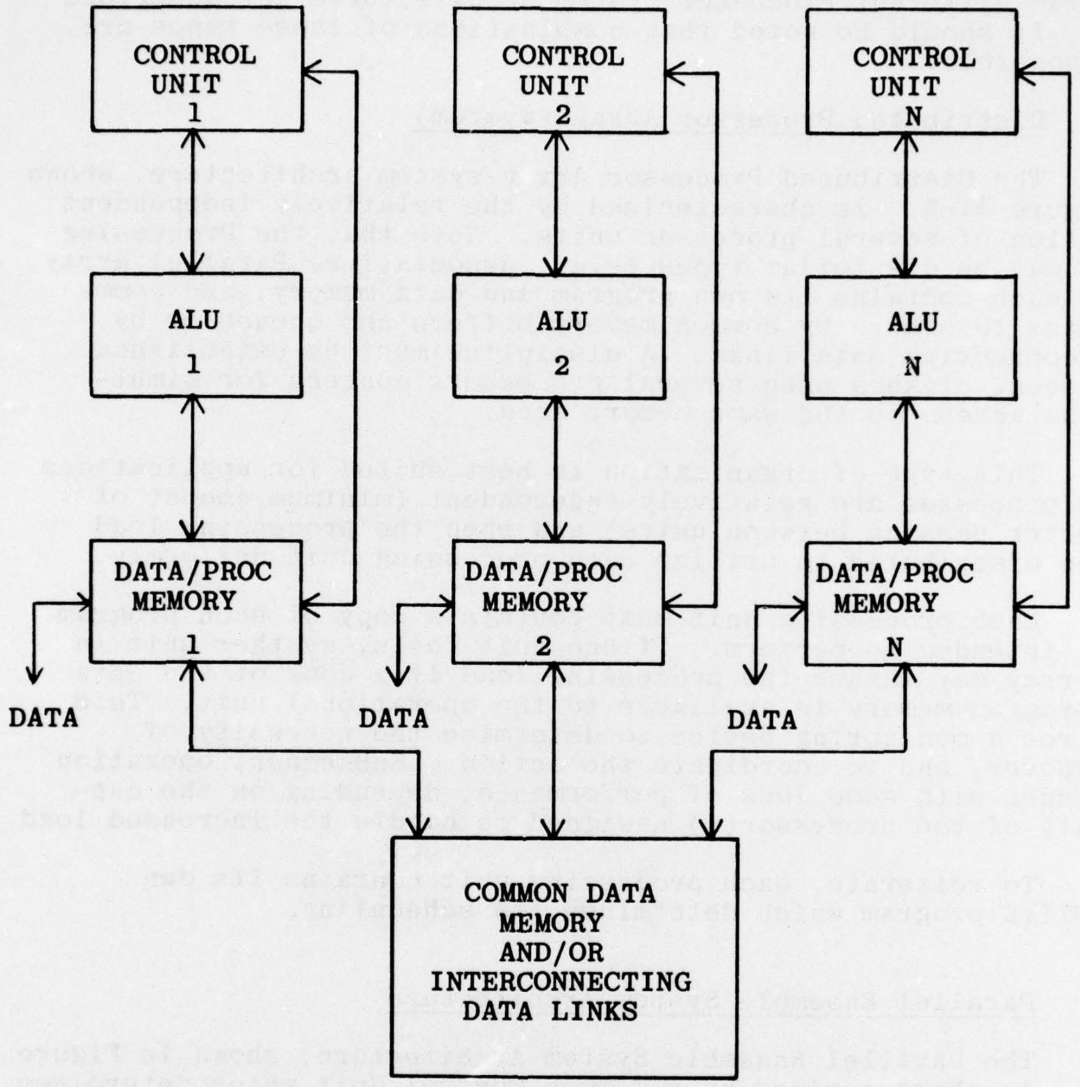
Each processing unit must contain a copy of each program it is intended to perform. If one unit fails, another unit in the array may assume the processing load if a copy of the data and program memory is available to the operational unit. This requires a monitoring device to determine the necessity of switchover, and to coordinate the action. Subsequent operation continues with some loss of performance, depending on the capability of the processor(s) assigned to handle the increased load.

To reiterate, each processing unit contains its own EXECUTIVE program which determines its scheduling.

Parallel Ensemble System Architecture

The Parallel Ensemble System Architecture, shown in Figure AI-6, is characterized by a Common Control Unit which determines the utilization of each processing unit. A contrast to the Distributed Processor Array just described may be beneficial: The processing units for the Distributed Array are "specialists" each performing predefined functions; the processing units for the Ensemble are "generalists" performing whatever function the Control Unit happens to schedule at any moment. Note that the control unit itself may be implemented by a separate Processor Unit.

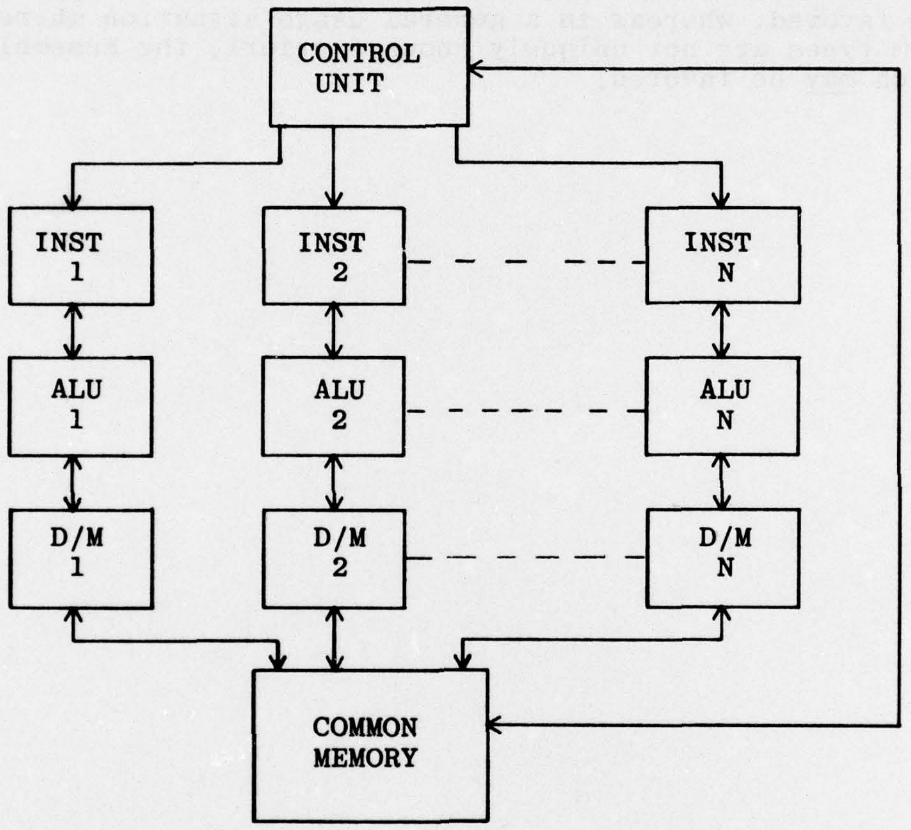
A high degree of resemblance exists between the Parallel Ensemble System block diagram and the Parallel Array Unit block



DISTRIBUTED PROCESSOR ARRAY (SYSTEM)

MULTIPLE INSTRUCTION STREAMS - MULTIPLE DATA STREAMS (MIMD)

FIGURE AI-5



PARALLEL ENSEMBLE SYSTEM ARCHITECTURE

MULTIPLE INSTRUCTION STREAM - MULTIPLE DATA STREAM (MIMD)

FIGURE AI-6

diagram. The difference is not the lack of common memory in the Unit architecture, for indeed a common memory could be utilized. The difference is that in the Parallel Ensemble System, actual machine level instructions are occurring asynchronously, with programs having been scheduled by the common control.

The main parameter in the choice between the Distributed System and the Ensemble System is the predictability, at the time of design, of the variation in loading between processing units. In a dedicated application, the Distributed approach may be favored, whereas in a general usage situation where problem types are not uniquely known a priori, the Ensemble approach may be favored.

OBSERVATIONS REGARDING COMMUNICATION PROCESSOR ARCHITECTURES

The Nature of the Functional Requirements for Communications

The overwhelming majority of different functions implemented in large scale circuit and data switching systems are statistically insignificant from the point of view of processing load. Thus, while these functions are important and must be implemented, they represent primarily a burden on program space rather than a processing burden. A properly optimized repertoire will allow the reduction of this program space, perhaps by as much as a factor of 50% but it is not likely that one can point to a particular functional requirement and note that it will affect the repertoire in such-and-such a manner. Most of the functional requirements have a diffuse impact on the architecture rather than a highly specific one. Changes are needed in address structure - this affects all functions and the need arises from all functions. A rich interrupt and channel structure is required, again a diffuse requirement in response to all functions. There are, however, some specifics which are needed in response to a few, readily identified requirements: code conversion instructions, jump table related instructions, possibly specialized instructions that are useful in performing path search routines and cyclic redundancy checks. Outside of these few areas, what is most needed are better means of implementing features already present in many computers in one form or another, and trying these features together to achieve a repertoire which is optimum for the communication task.

Specialized Non-Communication Requirements

Under this category, those features which are needed in any real-time system which must continue to work despite failures of its component elements are included. These features probably dominate the architecture more than the specifics of the communication task. A number of requirements stand out:

(1) Self-Failure Detection Mechanisms

Numerous facilities are required to detect and isolate malfunctions to the lowest automatically replaceable element. Included here are: illegal instruction traps, parity checks at interfaces, watchdog timers, illegal memory access attempt (assigned area violation), confidence checks, power transient interrupts.

(2) Automatic Bootstrap and Load Facilities

If fully automatic recovery is to be implemented, it cannot depend on manual bootstrapping, externally activated forced loading is mandatory.

(3) Configuration Switching and Element Isolation

The lowest automatically replaceable element (for example: computer, memory unit, device) must not only be isolated as a malfunctioning element but it must be possible to exit it from the configuration so that its malfunction will not poison the system, i.e., induce operational failures in otherwise functioning units. The ability to remove an element from the configuration and the ability to switch in a spare unit from a spares pool implies an extensive configuration control facility. This, in turn, implies inter-element interfaces that guarantee electrical and logical independence of the elements to a sufficiently high probability.

(4) Monitoring Devices

Monitoring facilities, whether implemented as specialized hardware or distributed software, are required to allow the detection of malfunctions where self-detection by the unit fails.

(5) Diagnostic Facilities

Since the system's availability is equally dependent on the MTBF and the MTTR, extensive diagnostic aids are required to reduce the unit down time once it has been placed in the hands of the maintenance technician and out of the working configuration. Such diagnostic facilities, be they implemented as hardware or software or a combination thereof, are essential.

Deficiencies of Present Systems

It was found in the applications studies that while most of the C/S systems utilized processors specifically designed for C/S control, the overwhelming majority of computers that have been used to date in a M/S communications role were not designed for that purpose. In most cases, they were adaptations of an existing or projected non-communication family or, because of a need for "product line" compatibility, they were heavily influenced by the companies existing or projected product line. Some features of commercial computers are merely unnecessary: predominant among these are floating point arithmetic, division, double precision arithmetic. Other features are only marginal utility; memory boundary violation, hardware relocation, virtual memory. The primary deficiencies of commercial practice follows:

(1) Rudimentary facilities related to recovery and configuration control. Most of the features described above are lacking or have been implemented in an ad-hoc manner as specialized designs. They have only been gradually incor-

porated into commercial systems. Generally, the communication field has led commercial practice by five to ten years in the items discussed above.

(2) Address space is insufficient - too little memory, not readily accessible.

(3) Dynamic storage allocation not implemented from the ground up. Where dynamic storage allocation is implemented, as in a virtual memory machine, the block size is too large. Furthermore, specialized facilities for indexing in dynamic memory are totally lacking.

(4) Not enough channels, interrupts and memory ports. The total number of devices (i.e. lines) that must be addressed in a typical communication system is far larger than that offered through the I/O repertoire of a typical commercial machine. Specialized devices must be constructed (usually on an ad-hoc basis) to allow access to the lines. This approach falls apart totally when DMA is needed for a large number of lines.

(5) Channel rate is wrong. Channel rate is almost always dictated by the device (e.g., disc or drum). Since these devices are made for commercial applications, their rates are often inappropriate. The problem more often than not is not that the device has too low a transfer rate, but that the transfer rate is too high.

(6) Growth structure is wrong. Since commercial practice is based on single computer configurations, growth in processing load is almost always met by increasing the size of the main computer, (and incidently, its back-up). Commercial practice is not oriented towards multi-computer complexes. They are relatively inflexible, with regard to system architectures, i.e., there is insufficient modularity

(7) Specialized instructions and interrupts missing. Instructions related to character detection and character sequence recognition are missing. Instruction indexing mode is limited to word indexing and character indexing. Bit manipulation instructions were added only relatively recently. Almost no facilities for extensive examination and manipulation of tables are provided.

APPENDIX II

USE OF CONTENT ADDRESSABLE MEMORY (CAM) IN COMMUNICATIONS SWITCHING

General

Hardware developments in the design of Content Addressable Memories (CAM's) have led to a promise of technical and economical feasibility for such devices in the next few years. In particular, the use of Large Scale Integration (LSI) has made possible approaches to CAM's that would have been unthinkably expensive a few years ago.

Concomitantly, theoretical studies on the utility of CAM's and the processors associated with them have been conducted over the past several years, to the point where the major theoretical problems attending their use have been analyzed and are now understood - while this is true for most areas, it is not true for all.

Historically, one of the major motivations for the development of CAM's was their application to a variety of large scale Information Storage and Retrieval (IS & R) problems, pattern recognition problems, and the like. In an IS & R system of any significant sophistication, a CAM offers the possibility of eliminating multi-threaded lists and the many ills to which they are heir. The application of CAM to such areas is generally considered proven, awaiting only technical economic availability.

To the extent that large scale IS & R becomes a function of message switching, CAM's are automatically applicable. However, it should be recognized that even the most complex retrieval functions within a message switch are in terms of bonafide information storage and retrieval systems, trivial. Even the foreseeable significant expansion of the IS & R functions of a message switch does not bring it into the realm of a full blown IS & R system. Should the switch be functionally expanded to include major IS & R processing, then the vast body of information pertaining to that application of CAM's would be automatically applicable to the message switch, and would not need yet another study on the subject. Therefore, this discussion is restricted to only those applications that are in the mainstream of message switching while it is still recognizable as being primarily a message switch, rather than an IS & R system that happens to do message switching on the side.

A Hypothetical CAM

There have been many proposed CAM memories, some built, others simulated. In order to lend some focus to our discussion, we shall propose a tentative CAM, its characteristics, etc. The

CAM concept discussed here is intended to be compatible with the general architecture of the CPS as discussed in Volume IV. Most of the characteristics have been taken from various Goodyear Associative Memories, as developed under contract with Rome Air Development Center.

Search Mode and Speed

A parallel, simultaneous search of all elements in the CAM in one memory cycle is assumed. A hardware implemented sequential search does not offer any significant advantage over doing the same thing with the instructions already in the repertoire. That is, using a masked search of a sub-element for a total or partial match on a specified key through N elements can be done in one instruction by non-CAM processing. Only if the hardware search is in parallel, by virtue of having comparison logic for every bit in the memory, would there be an advantage. Sequential search hardware does not represent a sufficient improvement in search time to warrant the expense.

What Is Searched, What Are The Keys?

Assume that the memory will manually provide the first (lowest order) element that satisfies the masked keys, where the key is specified as operating over a specified sub-element position and size within the element.

Instructions And Instruction Modes

Assume the following kinds of instructions.

Search Criteria

A logical search through a mask and a set of arithmetic searches for greater than, greater than or equal, equal, less than or equal, less than.

Boundary Control

A pair of fields that delimits the upper end and lower bounds in terms of normal memory locations over which the search is to take place.

Output Modes

(1) Address return, low.

The output is the lowest order element address that satisfied the search criteria.

(2) Address return, high.

The output is the highest order element address that satisfied the search criteria.

(3) Next address, low; next address high.

Depending on which is selected, the next highest (next lowest) address is returned. An address value is assumed to be contained in a specified register.

(4) Data mode read.

The same as the above four modes, except that a transfer to normal memory or register of the entire element is performed. The usual indexing and indirect modalities applies to the transfers as far as the receiving locations are concerned.

(5) Data mode write.

As above, except that a transfer is made into the block that was found by the search.

Note that 4 and 5 are actually MOVE instructions except that instead of having to specify explicitly both the source and the destination of the MOVE, one or the other is defined implicitly by the CAM operation. Such CAM operations would have to prevent the overwriting of the keys.

Key Index and Indirect

Indexing and indirect operations can be performed on the keys. In this mode, the indexing and indirect operation are identical to the usual indexing and indirect operations. The effective address calculation is made and the key brought forward. The found key is then used to activate the CAM operation.

CAM Index and Indirect

Indexing

The key value is incremented by the contents of the specified index register. The operation then proceeds in the normal manner.

Indirect

This could be very complicated and many possibilities abound. A simple case can be considered in which the CAM operation is performed and the element, less the keys that satisfies the search criteria, is brought up. The content of that element is used as an address to the data.

Post Indexing

The key is used to find an element that satisfies it. The contents of that element is used as an address, which is in turn modified by the contents of the specified index register. That effective address is used to perform the indicated read or write operation.

Jumps

A number of jump or skip instructions related to CAM operations would probably be implemented such as: JUMP (SKIP) on NO RESPONSE, EXACTLY ONE RESPONSE, MORE THAN ONE RESPONSE.

Normal Addressing

It is assumed that an element or its sub-elements can be addressed in the normal manner: that is all instructions of the normal computer still apply.

Size

Is there a size for a CAM below which it does not pay to use one? The answer is an overwhelming YES. If the CAM is small, say a few hundred words, the only way it can be used is by loading the table or data to be searched into the CAM area, and then applying the CAM operation to that table. A little reflection will show that this cannot pay off. Loading a table using a proper move instruction will take of the order of two memory cycles per word moved - one to extract the word from the location and one to move into the new location within the CAM. This assumes that there are powerful internal MOVE instructions. If there are no MOVE instructions, then the transfer memory cycles per word is even higher. Therefore, the surge time is proportional to 2N, where N is the number of words to be moved. If a binary halving technique is used to search a table, assuming 10 instructions per step, 20Log_2N is obtained for the search time in memory cycles. For this situation, the CAM loses for tables bigger than 64 words. If the processsing required for binary halving is doubled, it would merely double the point at which surging and hence CAM operation would not pay. It should also be recognized that binary halving searching is merely one of the more primitive search approaches. Given additional knowledge about the structure of the keys, their appearance probabilities, their expected spacing, and the relative usage frequencies, more elegant search techniques can be devised which bias things even further away from the use of a CAM. Consequently, it can be seen that a CAM would only be useful if it were available in large quantities. For example, the entire main memory is a CAM. Under this assumption, since numerous internal moves to use the CAM need not be made, the investigation into its applicability can be continued.

Potential Application Areas

The following are potential areas where the use of a CAM might be beneficial.

Routing Analysis

This operation, at present, takes of the order of 250 to 500 memory cycles per address. Most of this is taken up by validating and parsing the RI and not for searching the RI table. All of input processing consumes approximately 20% of the total processing load. RI related processing, for validation and finding the output line consumes less than 1% of the total processing load. The actual search is negligible.

Block Assignment and Release

This could be readily done with a CAM. The key would be set to a standard value, say 0, to indicate that the block in question was not in use. Part of the processing associated with finding or releasing a block is assuring that assignment and release will be done in a re-entrant fashion. This is needed since the assignment and release occurs at many priority levels. If this could be done in one instruction, part of this processing would be eliminated. However, the way this is normally done, is to keep a list of available space. The programmer knows exactly where to go to get the next block. If the programmer is not given LINK and UNLINK instructions, the process of linking and unlinking is tedious and the CAM has an advantage. However, given the LINK and UNLINK capabilities, the CAM has merely eliminated a few instructions by virtue of not having to have explicit queues.

Scanning Functions

These are periodic line monitoring functions whose purpose is to check every line in the system to see if certain conditions or situations have arisen. For example: has there been no activity on the line for more than a given time period? Has the elapsed time for a required response been exceeded? Approximately five to six percent of the processing in present message switching systems is consumed by this activity. With the proposed instruction repertoire, this will probably be cut down to less than 2 percent. A properly designed CAM would make this function statistically negligible.

General Housekeeping

Much of the extensive housekeeping now required to keep track of what phase of the process a message has gone thru becomes implicit if a CAM is used. The majority of internal queues disappear as explicit entities. For example, to start input processing, all block addresses that have not yet gone into input processing are requested. (Or all message header blocks by priority and time of arrival.) Explicit queues are not required. This application, through diffuse and hard to evaluate, probably represents the major application area of the CAM in a message switch. The primary benefit comes about not through the ability to rapidly search queues or their equivalent, but from the elimination of these elements as explicit items. An upper bound on the potential savings in processing time, under the new repertoire, is probably 20%.

Searches

One would expect that the primary application area for a CAM would be in search operations, particularly in the examination of queues. This would be true if a significant portion of the processing was involved in queue searching. However, it should be recognized that as message switching software has evolved, numerous techniques were developed whose primary purpose was to eliminate the explicit queue search. As it now stands, queue searches are statistically rare or conducted over relatively short queues - tens of items rather than thousands. As an example, consider the maintenance of a single output queue for all messages. If this were done, massive searches would be required for each message transmitted. However, a separate queue for each output line is normally maintained. In some systems, this may be further broken down to separate queues for different precedence or traffic types. Over-all, then, the organization has evolved in such a way as to eliminate the instances of searches in recognition of the fact that such operations are excessively time consuming. Eliminating the searches through the use of a CAM would provide some marginal performance improvement, but not the dramatic improvement one might have expected.

<u>Problems</u>

Generic Problems

A message switching system properly designed around the use of a CAM would probably be significantly different conceptually than those that are designed around more conventional machines. As can be seen from the above, the most promising difficulty with the use of CAM is the re-thinking of the organization of the software structure so as to obtain the maximum exploitation of the new technique.

Recovery Related Problems

Probably the most difficult areas regarding the use of a CAM are related to recovery. At present, dynamic tables, queues, line status information, message status information, etc. and other data which is needed in order to properly recover the traffic subsequent to a failure are stored, typically on a drum or disc, in an area called the "ledger". Ledgers are up-dated (or bench-marked) periodically, usually in multiple copies to protect against a ledger device failure. The elapsed tranfer time for writing the ledger copies is, at present, the single most time consuming part of the processing. The cycles stolen to transfer the ledger data are similarly the largest portion of the processing cycles over much of the system's operating range. Tables and queues, and other data to be ledgered are organized, typically, into ledgered and non-ledgered areas so that the ledger writing can be done efficiently. That is, while processing logic may dictate that items A, B, C, and D should be contiguous, if B and C need not be ledgered, it would be found that A and D were contiguous in one table, while B and C were stored in another, separate table. No small part of the design is centered about the attempt to eliminate or reduce internal data moves for ledgering. Typically, the ledgered data is concentrated and the various processing routines work out of those contiguous areas. When the time comes to ledger, it is relatively easy to lock out access to those tables.

In a CAM switch, most of the information would be stored in the data blocks, or message control blocks, and accessed by appropriate keys rather than by address. To make the most effective use of the CAM, all data pertaining to a given processing entity, such as line or a message, should be contained in the same line control block or message control block in the CAM. This means that ledgered and unledgered data are thrown together. This in turn means either accepting a large number of internal moves to concentrate the ledgered data, or ledgering all data, which again increased ledger related processing. Alternatively, one could distinguish between ledgered and unledgered data, storing the unledgered data in a CAM and the ledgered in a normal mode. But this would mean that all the extensive address calculations that would have been eliminated through the use of the CAM must be done for the ledgered data.

What it boils down to is the following paradox:

The use of a CAM makes much of the processing which had been explicit, implicit. Ledgered data, if it is to be done efficiently must be explicitly addressable - the act of making that data explicitly available erodes the effectiveness of the CAM.

Conclusions

The use of a CAM offers marginal advantages in most areas and potential advantages in others, but the degree is not clear cut. A 30% reduction in processing time is probably an optimistic upper bound on the total processing saved. The unsolved problems relating to the effective and non-trivial use of CAM's in a message switch are sufficient to make an accurate assessment of the cost-effectiveness of CAM's totally unknown. The implications of the use of CAM's in a message switch are such that a completely different software organization, based on CAM's would have to be examined before a definite conclusion could be reached. Measures of effectiveness based on other applications, such as information storage and retrieval, do not apply to message switching.

APPENDIX III

GLOSSARY OF TERMS AND ACRONYMS

AC	Alternating current; when used to describe a form of signaling it means that voice frequency tones or digital codes are used.
ACF	Administration Control Function. In a communications system it refers to the operations performed to administer and control the system. Refer to Volume II, Section 2.
ACK	An abbreviation for Acknowledge; a communications control character transmitted by a receiver as an affirmative response to a sender.
ACP-127	One of the two standard data formats used in AUTODIN (see also JANAP-128).
Ad hoc	(Latin) which means "for this case alone" or "special".
ADP	Automatic Data Processing.
Address Signaling	The part of the signaling procedure which indicates to the switching system the address of the called party.
Algorithm	Any species of notation used for calculations.
ALU	Arithmetic Logic Unit - A set of logic gates which performs the logical operations in a processor unit.
AMME/ATTC	Automated Multi-Media Exchange portion of the Automated Telecommunications Center. Refer to Volume II, Section 1.
Amorphous Uncrystalized	Without developed structural organization.
AN/TTC-25	A tactical military circuit switching system. Refer to Volume II, Section 1.
AN/TTC-39	A tactical military circuit and message switching system. Refer to Volume II, Section 1.
ARPA	Advanced Research Projects Agency. A high-level digital communications network. Refer to Volume II, Section 1.
ARPANET	(See ARPA).

ARQ	Automatic Request-Repeat. A mode of error control in which the receiving terminal is arranged to detect transmission errors and automatically transmits the request-repeat (RQ) signal to the transmitting terminal.
ASCII	American Standard Code for Information Interchange. Also called USASCII. This code was made a Federal Standard on 11 March 1968.
ASF	Address Signaling Function. In a circuit switch, it refers to the operations performed by the switching system to detect (or forward) the called party's telephone number. Refer to Volume II, Section 2.
Associative Array	Refers to a particular class of parallel array processor architectures.
ATR	Address Trap Register.
AUTODIN	Automatic Digital Information Network. Refer to Volume II, Section 1.
AUTOVON	Automatic Voice Network. Refer to Volume II, Section 1.
Avg.	Abbreviation of average.
Battery Feed Circuit	A circuit located in the circuit switching system which provides power to the telephone.
Baud	The unit of modulation rate. One baud corresponds to a rate of one unit interval per sec.
Baudot	A code used to digitize alphanumerics.
BCD	Binary Coded Decimal.
BCH	Base-Chaudhuri-Hocquenghem - a class of cyclic error detecting and correcting codes.
BH	Abbreviation for Busy Hour. In telephony, it refers to that hour of the day when the switching system is handling the most calls. The term is used generally to describe a system's maximum per hour throughput capacity.
Bit	A single binary element.

Bipolar	A term used by the transistor and integrated circuit industries to refer to a specific manufacturing technology.
Blocking	In message switching, it means to organize a (binary) message into modular sections or blocks consisting of a predefined number of bits or characters. In switching matrix design it means a condition which exists in a matrix whereby a path between two matrix ports cannot be established dur to existing conflicting paths.
BORAM	Block Oriented Random Access Memory
Bubble Memory	A binary memory device in which the storage mechanism is the presence or absence of a "bubble", a small, cylindrical, magnetic domain, at predetermined bit positions.
Byte	A set of eight bits (a character)
CAD	Computer Aided Design
CAM	Content Addressable Memory
CAMA	Centralized Automatic Message Accounting
CBU	Conference Bridge Unit. A circuit which allows the speech signals of three or more subscribers to be mixed in a strictly controlled manner such that each subscriber receives the transmissions of all other conferees but not his own.
CCD	Charge Coupled Device. A memory technology in which minority carrier charges are stored in "potential wells" created at the surface of the semiconductor.
CCF	Compatibility Conversion Function. In a message switch it refers to the operations performed to convert the received codes, formats, etc. of a message to those which are compatible with the internal processing elements of the system and/or with the receiving equipment. Refer to Volume II, Section 2.

CCITT	International Telegraph and Telephone Consultative Committee. An international organization which is concerned with the problems of the interworking of national communications systems. The acronym comes from the French name of the Committee.
Char	Abbreviation for character which is a set of bits used to encode a specific alphanumeric or control function.
Chip	Refers to a small, thin, usually square piece of semi-conductor material on which transistors of various types have been formed and interconnected to produce a specific operating function.
CID	Charge Injection Device. A memory technology simular to the charge coupled device.
CIF	Communications Interface Function. In message switching systems, it refers to the operations performed by the switch to coordinate the transmission and reception of digital data. Refer to Volume II, Section 2.
Classmark	A set of information which defines exactly the characteristics, signaling conventions, features, restrictions, etc., of a circuit switch terminal.
Clock	As used with respect to digital systems, it refers to a cyclic set of binary pulses used to cause digital operations to occur synchronously.
CMOS	Complimentary Metal-Oxide Semiconductor. An integrated circuit technology in which N-channel and P-channel MOS/FET's are used in a complimentary arrangement.
Code Switch	An electro-mechanical space division switching device used in circuit switching systems.
Comma Free Codes	A set of codes used to perform address and supervision signaling with digital end-instruments, characterized by the fact that frame synchronization is not required to achieve unambiguous recognition.

Common Channel	A system whereby all signaling for a number of communication paths is carried over one common channel.
Compelled	A form of trunk signaling in which the receiving station must answer each incoming signaling element with a specific (and usually) different signaling element.
COMSEC	Communications Security
Connect Map	A memory system which maintains a record of all existing communication paths in a switching matrix.
Core	A term used somewhat synonymously with memory due to the fact that most circuit and message switches use magnetic-core memories.
CP	Central Processor. Refers to a set of processor units complete with I/O channels, processor monitor units, control panels and some form of data and program memory.
CPF	Call Processing Function. In a circuit switch it refers to the operations performed by the system to coordinate the activities of the other functions and to perform routing. Refer to Volume II, Section 2.
CPS	Communications Processor System.
CPU	Communications Processor Unit.
CRT	Cathode Ray Tube. The "picture" tube used in oscilloscopes, Visual Display Units, etc.
Crosspoint	A switching point in a matrix. It may consist of any number of contacts operated in parallel.
Crypto Gear	Equipment specially designed to encrypt and decrypt voice and data communications.
C/S	Circuit Switch. A telephony switching system.
C.T.N.E.	A packet switching network. Refer to Volume II, Section 1.

CTU	Control Unit - part of the CPU.
CU	Channel Unit.
Curie Temperature	The temperature at which, in a particular ferro-magnetic substance, the phenomena of ferro-magnetism disappears and the substance becomes merely paramagnetic.
DAC	Direct Access Control.
DC Closure	A form of DC supervision signaling in which the circuit switch recognizes OFF-Hook by an increase in current flow due to the completion of the loop circuit caused by the DC closure.
Digit	When used with respect to address signaling, it refers to a specific single integer number.
Directory Table	A table which contains the telephone numbers of all subscribers homed on the switch.
Distribution Frame	A hardware device used as a terminal point for lines and trunks.
D/M	Abbreviation for Data Memory.
DMA	Direct Memory Access. A data transfer system in which the data memory serves as a common point between processor units or between separate programs within a processor unit.
DOT	Domain Tip. A form of bubble memory.
Doublet	A set of two bits.
D/P	Abbreviation for Data and Program Memory.
DP	Dial Pulse. The transmission of address signaling information by the mementary opening and closing of a DC circuit a specified number of times corresponding to the decimal digit which is dialed.
DPI	Data Processing Installations.
D/T	Abbreviation for Data Transfer control.

DTL	Diode Transistor Logic.
DTMF	Dual Tone Multi-Frequency. A method of signaling in which a combination of two frequencies out of a possible eight are used to transmit numerical address information.
E	Symbol for Erlang. The international dimensionless unit of traffic intensity.
E & M	A signaling arrangement characterized by the use of separate paths for signaling and the communication signals.
EBAM	Extended Block Addressable Memory.
ECL	Emmiter Coupled Logic. A semiconductor Logic device family technology.
E/D	Abbreviation for Enable/Disable.
EFL	Emitter Follower Logic.
e.g.	Exampli gratis (Latin) which means "for example" or "such as".
EMI	Electro-Magnetic Interference. Usually refers to noise or transcients caused by external equipment or natural phenomena.
EOM	End of Message. A communication control character used to indicate the end of a message.
etc.	Et cetera (Latin) which means "and so forth".
ETS-4	Electronic Toll Tandem Switching System - 4 wire. Refer to Volume II, Section 1.
EVS	Electronic Voice Switching System. Refer to Volume II, Section 1.
EWS-1	Electronische Wechsel Systeme (Electronic Switching System) Refer to Volume II, Section 1.
EXL	Executive Logic.
FAA	Federal Aviation Agency.

FAA/AFTN	Refer to Volume II, Section 1.
FAA/WMSC	Refer to Volume II, Section 1.
FET	Field Effect Transistor. A family of semi-conductor devices which are characterized by their ability to control current flow with the application of an electrostatic field.
Fixed Program	A set of logic gates interconnected to provide a specific set of logic operations which do not vary.
Folded Matrix	A class of switching matrices in which a path through the matrix enters and leaves the matrix on the same side by traversing the entire matrix twice.
Freq.	Abbreviation for Frequency.
Full Word	A set of 32 bits.
GOS	Grade of Service. A measure of the probability that, during a specific peak period of traffic, a call offered to a group of trunks or circuits will fail to find an idle path at the first attempt. Usually applied to Busy Hour Traffic.
G.P.	Abbreviation for General Purpose.
HAF	Header Analysis Function. In a message switch, it refers to the operations performed by the system to perform header validation and generation. Refer to Volume II, Section 2.
Half Word	A set of 16 bits.
Hamming	A class of error correcting codes.
Header	That portion of data message which contains all of the information concerning its disposition and handling requirements. Refer to Volume II, Section 2.
HEX	Hexidecimal.
HH	High High. Refers to packet switch tandem trunk traffic.

HLN	High Level Network. A digital network consisting exclusively of High High traffic.
Holding Time	The duration of occupancy of a traffic path by a call (or message). Sometimes used to mean the average duration of occupancy of one or more paths by calls (or messages).
Homed	Refers to subscribers or terminals whose telephone number or identification code has the same office number or code as the switching node.
H/R	Historical/Recovery. Refers to a degree of message accountability. Refer to Volume II, Section 2.
H/R/T	Historical/Recovery/Traffic. Refers to a degree of message accountability.
HSL	High Speed Line. A message traffic line or trunk which operates at 9600 bps.
$\overline{HT}$	Average Holding Time.
IC	Integrated Circuit. Refer to Volume VI.
ICCS	Integrated Communications Control System. Refer to Volume II, Section 1.
ID	Abbreviation for Identification.
i.e.	Id est (latin) which means "that is".
ICMS	Integrated Circuit and Message Switch.
I^2L(ILL)	Integrated Injection Logic. A semi-conductor technology used to produce high speed, low power IC logic devices.
I/O	Input/Output
Information Signaling	The set of tones used by a circuit switch to inform the subscriber of the calls' progress. e.g. ringback tone.
INST	Abbreviation for Instruction.
IS&R	Information Storage and Retrieval.

ITA#2	International Teletypewriter Alphabet #2. A specific character code used extensively in AUTODIN.
ITS	In Transit Storage. One of the memory storage requirements of a message switch.
JANAP-128	One of the standard data formats used in AUTODIN.
J-FET	Junction Field Effect Transistor.
LAMA	Local Automatic Message Accounting.
LCD	Liquid Crystal Display.
LED	Light Emiting Diode.
LDMS/ NAVCOMPARS	Refer to Section 2, Paragraph 2.3.1.11.
LIF	Line Interface Function. Refer to Section 2, Paragraph 2.2.2.1.
Line	The communication path between a subscriber's (or user's) end instrument and the switching system.
Linemark	A set of information which defines exactly the characteristics, code, mode, format, features, restrictions, etc. of a message switch terminal.
Link	Used to describe the interconnection paths between switching systems. In matrix design, it refers to the paths connecting switching arrays.
LKG	Line Key Generator. One of the TENLEY family of equipments.
L-L	Abbreviation for Local to Local, a class of circuit switch traffic.
LLN	Low Level Network.
LMF	Language Media Format. In message switching, a set of formats used to provide compatibility with a variety of terminal equipments.

LSB	Least Significant Bit.
LSI	Large Scale Integration. Refers to an integrated circuit with 300 or more logic gates or their equivalent.
LSL	Low Speed Line. A data line operating at baud rates of 600 bps or less.
L-T	Abbreviation for Line to Trunk, a class of circuit switch traffic.
Magnetic Bubble	See Bubble Memory.
MCF	Maintenance Control Function. In a communication system it refers to the operations performed to monitor and maintain the system. Refer to Volume II, Section 2.
MF	Multi-Frequency. A method of transmitting address information in which the identity of each of the ten possible digits plus the required supervision functions is determined by a combination of two out of six possible frequencies.
MIMD	Multi-Instruction - Multi-Data. Refers to a class of processor architectures.
MISD	Multi-Instruction - Single Data. Refers to a class of processor architectures.
MOS	Metal Oxide Semiconductor. Refers to a family of FET's.
MOS/FET	Metal Oxide Semiconductor/Field Effect Transistor (See MOS above).
MPF	Message Processing Function. In a message switch. it refers to the operations performed by the system to achieve routing and system control. Refer to Volume II, Section 2.
M/S	Message Switch.
MSB	Most Significant Bit.

MSI	Medium Scale Integrated.
Msg.	Abbreviation for Message.
MSL	Medium Speed Line. A data line operating at a speed from 1200 to 4800 bps.
MTBF	Mean Time Between Failures.
MTTR	Mean Time To Repair.
Multi-word	A set of two or more full words.
MXL	Matrix Logic.
NACK	Negative Acknowledge. A communications control character transmitted by a receiver as a negative response to the sender.
NCF	Network Control Function. Refer to Volume II, Section 2.
N-Channel	A type of FET in which N type material is used in the channel (current conductor).
N/D	Narrative/Data. Terms used to distinguish two types of message traffic.
Nibble	A set of 4 bits.
NMF	Network Management Function. Refer to Volume II, Section 2.
Numbering Plan	A plan by which telephone numbers, services, features, etc. are assigned unambiguously in a telephony system.
NRL	Non-Register Logic.
NX-1E	Refer to Volume II, Section 1.
OP-Code	Abbreviation for Operand Code.
Order-Wire	A circuit for use by maintenance personnel for communication.
PC	Program Counter.

PCF	Path Control Function. In circuit switching, it refers to the operations necessary to create and control real-time communication paths. Refer to Volume II, Section 2.
P-Channel	A type of FET in which P type material is used in the channel (current conductor).
PE	Processing Element synonymous with ALU.
PLA	Plain Language Address.
PMF	Position Management Function. In a communication system it refers to the operation necessary to control the man/machine interfaces. Refer to Volume II, Section 2.
PMOS	P-Channel MOS
PNPN	Abbreviation for a four-layer diode (a two terminal device) in which P indicates P-material and N indicates N-material.
Pooled	When used in reference to equipment in a switching system, it means that any unit from the pool of identical units may be used when necessary.
PROC	Abbreviation for Process.
PTC	Pentagon Telecommunications Center. Refer to Volume II, Section 1.
RAM	Random Access Memory.
RAS	Random Access Storage.
RGL	Register Logic.
RMF	Remote Maintenance Function. Refer to Volume II, Section 2.
RTL	Resistor Transistor Logic.
R.I.	Routing Indicator. A set of characters in a data message which indicates the destination of the message.

ROM	Read Only Memory.
SAMSON	Strategic Automatic Message Switching Operational Network. Refer to Volume II, Section 1.
SATIN IV	Refer to Volume II, Section 1.
SCL	Scanning Logic.
SCR	Silicon Controlled Rectifier - a four layer diode with a gate control input making it a three terminal device.
SF	Single Frequency. A method of signaling in which a single tone (2600 Hz for example) is placed on the communication path.
S/F	Store and Forward. A class of message switches.
SIMD	Single Instruction - Multiple Data. Refers to a class of processor architectures.
SISD	Single Instruction - Single Data. Refers to a class of processor architectures.
SOM	Start of Message. A control character indicating the start of a message.
SOS	Silicon on Saphire. A semiconductor technology.
SOW	Statement of Work.
SSF	Supervision Signaling Function. In a circuit switch, it is the operations performed by the system to detect or send supervision signals. Refer to Volume II, Section 2.
SWIFT	Refer to Volume II, Section 1.
Sync	Abbreviation for synchronize. Also appears as synch.
TADSS	Tactical Automatic Digital Switching System. Refer to Volume II, Section 1.
TARE	Telegraphic Automatic Relay Equipment. Refer to Volume II, Section 1.

T & C	Timing and Control.
TCF	Transmission Control Function. Refer to Volume II, Section 2.
TDM	Time Division Multiplex.
TELEX/WUI	Refer to Volume II, Section 1.
TENLEY	The name given to a family of communications security equipment.
Translation	In telephony, the operation of converting the dialed digits into matrix port identifications.
TRAP	Programmable Hardware Monitoring System.
Tri-State	Refers to the output circuits used in certain logic devices in which three states are possible; a low impedance to the minus supply, a low impedance to the plus supply, and a high (or open) impedance to either supply.
TRL	Translator Logic.
Trunk	A communication path between switching centers.
T-T	Abbreviation for Trunk to Trunk, a class of circuit switch traffic.
T^2L(TTL)	Transistor-Transistor Logic. A family of integrated circuit logic devices.
UCALU	Universal Communications Arithmetic Logic Unit.
VDU	Visual Display Unit.
VH	Abbreviation for Very High. Refers to message line speeds from 19.6 kbps on up.
VHD	Abbreviation for Very High speed line, Dedicated.
VHL	Abbreviation for Very High speed Line.
VHS	Abbreviation for Very High speed line, Switchable.

Vis-a-vis	French for "face-to-face". Means "Opposite". Occasionally used to mean "with respect to".
Watchdog	When used with the words "timer" or "timing", it refers to a timer which is keeping track of how long a program, unit, etc. of the system takes to complete a specific task.
Wired-Logic	A set of logic circuits interconnected to perform a fixed set of functions.
XOR	Abbreviation for "Exclusive OR".

APPENDIX IV

BIBLIOGRAPHY

MODELING

Fischer, M.J., and Harris, T.C., "A Model for Evaluating the Performance of an Integrated Circuit and Packet-Switched Multiplex Structure", *IEEE Transactions on Communications*, Vol. COM-24, No. 2, February 1976, pp. 195-202.

Gordon, Geoffrey, "System Simulation", *Prentice-Hall*, Englewood Cliffs, N.J., 1969.

Kleinrock, L., "Analytic and Simulation Methods in Computer Network Design", *Spring Joint Computer Conference*, 1970, pp. 569-579.

Kleinrock, L., "Models for Computer Networks", *School of Engineering and Applied Science, University of California*, Los Angeles, California, pp. 21-9 - 21-16.

Meister, B., H.R. Miller, and H.R. Rudin, Jr., "Optimization of a New Model for Message-Switching Networks", *IBM Zurich Research Laboratory*, pp. 39-16 - 39-21.

SWITCHING NETWORKS

Abramson, N., "The Aloha System - Another Alternative for Computer Communications", *Fall Joint Computer Conference*, 1970, pp. 281-285.

Baran, P., "On Distributed Communications Networks", *IEEE Transactions on Communications Systems*, March 1964, pp. 1-9.

Carr, C.S., S.D. Crocker, and V.G. Cerf, "HOST-HOST Communication Protocol in the ARPA Network", *Sprint Joint Computer Conference*, 1970, pp. 589-597.

Cederbaum, I., and I. Paz, "On Optimal Routing Through Communications Nets", *IEEE Transactions on Communications*, August 1973, pp. 936-941.

Chou, W., and H. Frank, "Survivable Communication Networks and the Terminal Capacity Matrix", *IEEE Transactions on Circuit Theory*, Vol. CT-17, No. 2, May 1970, pp. 192-197.

Crocker, S.D., J.F. Heafner, R.M. Metcalfe, and J.B. Postel, "Function-Oriented Protocols for the ARPA Computer Network", *Spring Joint Computer Conference*, 1972, pp. 271-279.

Despres, R.F., "A Packet Switching Network with Graceful Saturated Operation", *ICCC*, 1972, pp. 345-351.

Dickson, P.A., "ARPA Network Will Represent Integration on a Large Scale", *Electronics*, September 1968, pp. 131-134.

Farber, D.J., "Networks: An Introduction", *Datamation*, April 1972, pp. 36-39.

Fear, J.L., CDR, "United States Coast Guard Communications, A Systems Approach for the Enhancement of Maritime Safety", *Telecommunications Management Division*, U.S. Coast Guard Headquarters, Washington, D.C., pp. 7D-1 - 7D-5.

Fields, J.S., "Multicommodity Throughput in Digital Data Networks with Finite Storage", *IEEE Transactions on Communications*, July 1973, pp. 836-842.

Frank, H., and I.T. Frisch, "Analysis and Design of Survivable Networks", *IEEE Transactions on Communication Technology*, Vol. COM-18, No. 5, October 1970, pp. 501-519.

Frank, H., M. Gerla, and W. Chou, "Issues in the Design of Large Distributed Computer Communication Networks", *Network Analysis Corporation*, Glen Cove, New York, pp. 37A-1 - 37A-9.

Frank, H., I.T. Frisch, and W. Chou, "Topological Considerations in the Design of the ARPA Computer Network", *Spring Joint Computer Conference*, 1970, pp. 581-587.

Franks, R.L., and R.W. Rishel, "Optimum Network Call-Carrying Capacity", *The Bell System Technical Journal*, Vol. 52, No. 7, September 1973, pp. 1195-1214.

Fultz, G.L., and L. Kleinrock, "Adaptive Routing Techniques for Store-and-Forward Computer Communication Networks", *Computer Science Department, University of California*, Los Angeles, pp. 39-1 - 39-7.

Gimpelson, L.A., "Network Management: Design and Control of Communications Networks", *Electrical Communication*, Vol. 49, No. 1, 1974, pp. 4-23.

Herzog, U., "Message-Switching Networks with Alternate Routing", *ITC 7*, pp. 415/1 - 415/8.

Inose, H., "Communication Networks", *Scientific American*, September 1972, pp. 117-128.

Kleinrock, L., "Scheduling, Queueing, and Delays in Time-Shared Systems and Computer Networks", *Computer Science Department, University of California*, Los Angeles, pp. 95-141.

Lorcher, W., "Point-to-Point Selection in Link Systems", *University of Stuttgart, ITC 7*, pp. 317/1 - 317/8.

McKenzie, A.A., B.P. Cosell, J.M. McQuillan, M.J. Thrope, "The Network Control Center for the ARPA Network", *ICCC*, 1972, pp. 185-191.

McQuillan, J.M., W.R. Crowther, B.P. Cosell, D.C. Walden, and F.E. Heart, "Improvements in the Design and Performance of the ARPA Network", *Fall Joint Computer Conference*, 1972, pp. 741-754.

Penisten, G.E., "Telecomputing System Concept Will Help Us Reach New Levels of Productivity", *Communications News*, February 1975.

Pierce, J.R., "Communication", *Scientific American*, September 1972, Vol. 227, No. 3, pp. 31-41.

Raymond, H.G., "A Queueing Theory Approach to Communication Satellite Network Design", *TRW Systems Groups*, Redondo Beach, California, pp. 42-26 - 42-31.

Roberts, L.G., and B.D. Wessler, "The ARPA Network", *Computer Communications Network*, Prentice-Hall, 1972, pp. 485-501.

Rubin, I., "An Approximate Time-Delay Analysis for Packet-Switching Communication Networks", *IEEE Transactions on Communications*, Vol. COM-24, No. 3, February 1976, pp. 210-221.

Sanchis, B.F., and J.E. Villar de Villacian, "Formulation of Blocking Functions for Network Graphs", *Electrical Communication*, Vol. 47, No. 4, 1972, pp. 239-244.

Schulz, K., Fernmeldetechnisches Zentralamt, "The Use of Service Computers in Stored-Programmed Controlled Telephone Exchange (EWS-1) Networks of the Deutshe Bundespost", *ISS Symposium Record*, 1974.

Slabon, R., Fernmeldetechnisches Zentralamt, "Introduction and Realization of the PCM Switching Network in the EWS-1 Electronic Switching System", *ISS Symposium Record*, 1974.

Sloane, N.J.A., "On Finding the Paths Through a Network", *The Bell System Technical Journal*, Vol. 51, No. 2, February 1972, pp. 371-390.

Verma, P.K., and A.M. Rybczynski, "The Economics of Segregated and Integrated Systems in Data Communication With Geometrically Distributed Message Lengths", *IEEE Transactions on Communications*, November 1974, pp. 1844-1848.

Wright, E.P.G., "Error Correction: Relative Merits of Different Methods", *Electrical Communication*, Vol. 48, No. 1 and 2, 1973, pp. 134-145.

Zadeh, N., "On Building Minimum Cost Communication Networks", *Thomas J. Watson Research Center, IBM*, Yorktown Heights, New York, pp. 315-331.

<u>SWITCHING SYSTEMS</u>

Amstutz, S.R., "A New Store and Forward Message Switching System", *IEEE Transactions on Communications Technology*, August 1971, pp. 528-529.

Aro, E., T.K. Cheney, J.R. McMullen, and C.B. Nennerfelt, "NX-1E, an Electronically Controlled Crossbar Switching System", *IEEE Transactions on Communication Technology*, Vol. COM-18, No. 6, December 1970, pp. 734-740.

Aro, E., E.A. Lissakers, and G.D. Southard, "Processor-Controlled Office Testing System", *North Electric Company, Paul H. Henson Research Center*, Columbus, Ohio, 1975.

Beesley, J.H., "Practical Multistage Space-Time Switching Networks", *IEEE Transactions on Communications*, August 1973, pp. 919-922.

Belton, R.C., and J.R. Thomas, Post Office Telecommunications Headquarters, "The United Kingdom Post Office Packet Switching Experiment", *ISS Symposium Record*, 1974.

Dankowski, J.J., G.F. Dooley, and S.Y. Persson, "ETS-4 System Overview", *North Electric Company, Paul H. Henson Research Center*, Columbus, Ohio, 1975.

Field, R.G., GTE Sylvania Inc., "Techniques for Synchronous Time Division Digital Switching", *ISS Symposium Record*, 1974.

Fritz, R.A., Siemens Aktiengesellshaft, "EWS Support Software", *ISS Symposium Record*, 1974.

Groenendaal, G.C., L.O.G. Kristiansson, and J.B.H. Peek, "The Behavior of a Hold-and-Forward Concentrator for a Type of Dependent Interarrival Statistics", *IEEE Transactions on Communications*, Vol. COM-22, No. 3, March 1974, pp. 273-278.

Hagstrom, K.L., "Non-Symmetric Solid State Four Stage Switching Matrix", *ISS*, 1974, pp. 218/1 - 218/9.

Hassell, G.W., and R.D. Packard, "Load and Network Control in ETS-4", *North Electric Company, Paul H. Henson Research Center*, Columbus, Ohio, 1975.

Heart, F.E., R.E. Kahn, S.M. Ornstein, W.R. Crowther, and D.C. Walden, "The Interface Message Processor for the ARPA Computer Network", *Spring Joint Computer Conference*, 1970, pp. 551-567.

Howe, R.G., "LDMX/NAVCOMPARS Systems", *SIGNAL*, February 1974, pp. 8-10.

Kuczura, A., and S.Y. Persson, "ETS-4 Processing Capacity and Load Testing", *North Electric Company, Paul H. Henson Research Center*, Columbus, Ohio, 1975.

Lissakers, E.A., and D.P. Oestreich, "Multiprocessor Control of ETS-4", *North Electric Company, Paul H. Henson Research Center*, Columbus, Ohio, 1975.

Maiwald, D., H.R. Mueller, H.R. Rudin, and C.H. West, "Integrated Communication System Performance", *IBM Zurich Research Laboratory*, pp. 24-13 - 24-21.

Marcus, M.J., "Bounds for the Growth of the Number of Contacts in Connecting Networks", *United States Air Force, ITC 7*, pp. 315/1 - 315/7.

Moses, H.J., and D.L. Wagers, "ETS-4 System Verification Testing", *North Electric Company, Paul H. Henson Research Center*, Columbus, Ohio, 1975.

Pollen, L.K., GTE Sylvania Inc., "A Tactical Military Electronic Central Office System Organization", *ISS Symposium Record*, 1974.

Robbins, J.B., Major Gen., USAF, "The Convergence of Computer and Communication Technologies", *SIGNAL*, January 1975, pp. 21-23.

Roberts, L.G., "Computer Report III - Data by the Packet", *IEEE Spectrum*, February 1974, pp. 46-51.

Tyndall, E.G., "Computer Controlled Telex", *RCA Corporation*, Camden, New Jersey, pp. 10C-1 - 10C-4.

Watts, J.A., "NX-1E Routing Control Complex", *IEEE Transactions on Communication Technology*, Vol. COM-18, No. 6, December 1970, pp. 729-733.

COMPUTER COMPLEXES AND SYSTEMS

Batcher, K., "STARAN Parallel Processor System Hardware", *AFIPS Volume 43*, 1974.

Beizer, B., "The Architecture and Engineering of Digital Computer Complexes", *Plenum Press*, Chapter 7, Vol. 1, New York-London, 1971.

Berenyi, I., "Computers in Eastern Europe", *Scientific American*, October 1970, pp. 102-108.

Berg, R.O., H.G. Schmitz, and S.J. Nuspl, "PEPE - An Overview of Architecture, Operation, and Implementation", *Honeywell Systems and Research Center*, St. Paul, Minnesota, pp. 312-317.

Bergland, G.D., "Fast Fourier Transform Hardware Implementations - A Survey", *IEEE Transactions on Audio and Electro-Acoustics*, Vol. AU-17, No. 2, June 1969, pp. 109-119.

Bonseigneur, P., "Description of the 7600 Computer System", *Computer Group News*, May 1969, pp. 11-15.

Bowdon, E.K., Sr., "Priority Assignment in a Network of Computers", *IEEE Transactions on Computers*, Vol. C-18, No. 11, November 1969, pp. 1021-1026.

Chuang, H., and S. Das, "An Approach to the Design of Highly Reliable and Fail Safe Digital Systems", *AFIPS Volume 43*, 1974.

Comfort, W.T., "A Computing System Design for User Service", *Proceedings of FJCC*, 1965.

Dennis, J.B., "Segmentation and the Design of Multi-Programmed Computer Systems", *Journal of ACM*, Vol. 12, No. 4, 1965.

Feistel, H., "Cryptography and Computer Privacy", *Scientific American*, Vol. 228, No. 5, May 1973, pp. 15-23.

Feldman, J., and L. Fulmer, "RADCAP - An Operational Parallel Processing Facility", *AFIPS Volume 43*, 1974.

Frank, H., R.E. Kahn, and L. Kleinrock, "Computer Communication Network Design - Experience with Theory and Practice", *Spring Joint Computer Conference*, 1972, pp. 255-270.

Freiser, M.J., and P.M. Marcus, "A Survey of Some Physical Limitations on Computer Elements", *IEEE Transactions on Magnetics*, Vol. MAG-5, No. 2, June 1969, pp. 82-90.

Gunderson, D.C., "Some Effects of Advances in Memory System Technology on Computer Organization", *Computer*, November/December, pp. 7-11.

Jackson, P.E., and C.D. Stubbs, "A Study of Multiaccess Computer Communications", *Spring Joint Computer Conference*, 1969, pp. 491-504.

Jaye, F., and R. Riener, "Use of a Multiprogramming Mini-Computer for Real Time Control Applications", *AFIPS Volume 43*, 1974.

Kast, F.E., Rosenzweig, J.E., "Organization and Management: A System Approach", New York: *McGraw-Hill*, 1970.

Keyes, R.W., "Physical Problems and Limits in Computer Logic", *IEEE Spectrum*, May 1969, pp. 36-45.

Kuehn, R.E., "Computer Redundancy: Design, Performance, and Future", *IEEE Transactions on Reliability*, Vol. R-18, No. 1, February 1969, pp. 3-11.

Loyd, G., and A. VanDam, "Considerations for Micro-Programming Languages", *AFIPS Volume 43*, 1974.

MacSorley, O.L., "High-Speed Arithmetic in Binary Computers", *Proceedings of the IRE*, 1961, pp. 67-91.

Mitchell, J., C. Knadler, G. Lunsford, and S. Yund, "Multiprocessor Performance Analysis", *AFIPS Volume 43*, 1974.

Ornstein, S.M., F.E. Heart, W.R. Crowther, H.K. Rising, S.B. Russell, and A. Michel, "The Terminal IMP for the ARPA Computer Network", *Spring Joint Computer Conference*, 1972, pp. 243-254.

Powers, V., "An Analytical and Empirical Study of the Common Data Base Concept and Generalized Data Base Management Systems", 1971 *Univ. Microfilms*, Ann Arbor, Michigan.

Purhami, B., and A. Auizienis, "A Study of Fault Tolerance Techniques for Associative Processors", *AFIPS Volume 43*, 1974.

Ramamoorthy, C., "Pipelining - The Generalized Concept and Sequencing Strategies", *AFIPS Volume 43*, 1974.

Ramamoorthy, C., and H.F. Li, "Efficiency in Generalized Pipeline Networks", *AFIPS Volume 43*, 1974.

Roberts, L.G., "Resource Sharing Computer Networks", *Advanced Research Projects Agency*, Paper 7B.2, pp. 326-327.

Rubin, A.I., "Analog/Hybrid - What It Was, What It Is, What It May Be", *Fall Joint Computer Conference*, 1970, pp. 641-652.

Slotnick, D.L., "The Fastest Computer", *Scientific American*, February 1971, pp. 76-87.

Solomon, M.B., Jr., "Economies of Scale and the IBM System/360", *Communications of the* ACM, Vol. 9, No. 6, June 1966, pp. 435-440.

Steele, J.M., and R.C. Mattson, "Architecture of a Universal Communications Processor", *Computer Design*, November 1973, pp. 63-68.

Weitzman, C., "Optical Technologies for Future Computer System Design", *Computer Design*, April 1970, pp. 169-175.

Winograd, S., "How Fast Can Computers Add?", *Scientific American*, October 1968, pp. 93-100.

Winograd, S., "On the Time Required to Perform Addition", *Journal of the Association for Computing Machinery*, Vol. 12, No. 2, April 1965, pp. 277-285.

Wintz, P.A., and A. Habibi, "Fast Multipliers", *IEEE Transactions on Computers*, February 1970, pp. 153-157.

Wishner, H.D., "A Minimum Cost Computer for a 1975 Launch Vehicle", *Computer Design*, December 1969, pp. 41-46.

PLANNING AND STUDY DOCUMENTS

Baechler, D.O., "Aerospace Computer Characteristics and Design Trends", *Computer*, January/February, pp. 46-57.

Brandon, R.H., "Management Standards for Data Processing", *Van Nostrand*, Princeton, N.J., 1963.

Grombacher, G.S., Brig. Gen., "Communications Plans for SAFEGUARD", *SIGNAL*, October 1974, pp. 12-18.

GTE Sylvania Inc., Eastern Division, "SENET-DAX Study, Preliminary Study Review, Technical Summary", *GTE Sylvania Electronic Systems Group*, December 10-11, 1975.

Harlfinger, F.J., II, Vadm., "Command, Control and Communications in the Navy Now and In the Future", *SIGNAL*, February 1974, pp. 6-7.

Kummerle, K., "Multiplexor Performance for Integrated Line and Packet Switched Traffic", *The Second International Conference on Computer Communications Record, IBM*, Ruschlikon, Switzerland.

Lyons, R.E., Dr., and Dr. R.W. Parkinson, "System Design Considerations for the Future Defense Communications System", *Defense Communication Engineering Center*, Reston, Va., pp. 17D-1 - 17D-7.

"Overseas AUTOVON Network Switching Plan", *DCA Circular*, 370-V185-7.

Schlain, E.L., E. Dundatschek, F.M. McDonnel, and P.P. Boehm, "Integrated Circuit/Message Switch Feasibility Model Development, Test, and Evaluation", *Volume IV Common Communications Processor*, RADC-TR-72-27, November 1972, (905360).

Segerstrom, C.A., and B.L. Troutman, Jr., "Trends Toward Air Force Digital Information Systems", *Headquarters Electronics Systems Division*, L.G. Hanscom Field, Mass., and *The MITRE Corporation*, Bedford, Mass., pp. 17A-1 - 17A-8.

Turn, R., "Air Force Command and Control Information Processing Trends in Hardware Technology", R-10011-PR, October 1972.

Zabropolo, P., "Flexible Multiplexing for Networks Supporting Line Switched and Packet Switched Data Traffic", *The Second International Conference on Computer Communications, IBM*, Stockholm, August 12-14, 1974.

TRANSMISSION/TRAFFIC

Aghib, E.G., E.F. Sock, "Section IV: Introduction to Data Transmission in Europe", *Electrical Communication*, Vol. 48, No. 1 and 2, 1973, pp. 108-120.

BCR Staff, "Data Transmission Performance on the DDD Network", *Business Communications Review*, November/December 1972, pp. 15-21.

Busignies, H., "Communication Channels", *Scientific American*, September 1972, pp. 99-113.

Kusunoki, S., A. Furuya, and T. Masuda, "Analog Transmission Performance on the Switched Telecommunications Network", *Japan Telecommunications Review*, April 1974, pp. 96-104.

Pierce, J.R., C.H. Coker, and W.J. Kropfl, "An Experiment in Addressed Block Data Transmission Around a Loop", *Bell Telephone Laboratories, Inc.*, Murray Hill, New Jersey, Paper 4F.5, pp. 222-223.

MEMORIES

Arora, S.R., and A. Gallo, "Optimal Sizing, Loading, and Re-Loading in a Multi-Level Memory Hierarchy System", *Spring Joint Computer Conference*, 1971, pp. 335-336.

Bardos, A., "Wideband Holographic Recorder", *Applied Optics 13*, April 1974, p. 832.

Barrekette, E.S., "Trends in Storage of Digital Data", *Applied Optics 13*, April 1974, p. 749.

Bobeck, A.H., "A Second Look at Magnetic Bubbles", *IEEE Transactions on Magnetics*, Vol. MAG-6, No. 3, September 1970, pp. 445-446.

Bobeck, A.H., and H.E.D. Scovil, "Magnetic Bubbles", *Scientific American*, June 1971, pp. 78-90.

Bonyhard, P.I., I. Danylchuk, D.E. Kish, and J.L. Smith, "Applications of Bubble Devices", *IEEE Transactions on Magnetics*, Vol. MAG-6, No. 3, September 1970, pp. 447-451.

Brown, B.R., "Optical Data Storage Potential of Six Materials", *Applied Optics 13*, April 1974, p. 761.

Brown, J.R., Jr., "First - and Second-Order Ferrite Memory Core Characteristics and Their Relationship to System Performance", *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 4, August 1966, pp. 485-501.

Burke, H.K., and G. Michon, "CID's Charting a New Course", *Optical Spectra 8*, March 1974, p. 29.

Chang, H., "Capabilities of the Bubble Technology", *AFIPS Conference Proceedings 43*, 1974, p. 847.

Colton, D.R., "Charge Coupled Device Memories", *International Magnetics Conference in Toronto*, May 1974.

Cricchi, J.R., "MNOS - A Non-Volatile Semiconductor Storage", *International Magnetics Conference in Toronto*, May 1974.

England, W.A., "Applications of Plated Wire to the Military and Space Environments", *IEEE Transactions on Magnetics*, September 1970, pp. 528-534.

Gaylord, T.K., "Optical Memories", *Optical Spectra 8*, June 1974, p. 29

Greifer, A.P., "Ferrite Memory Materials", *IEEE Transactions on Magnetics*, Vol. MAG-5, No. 4, December 1969, pp. 774-811.

Guzik, R.P., "Optical Data Recording", *Electro-Optical System Design 6*, June 1974, p. 22.

Katzan, H., Jr., "Storage Hierarchy Systems", *Pratt Institute*, Brooklyn, New York, pp. 325-335.

Lee, T.C., "Holographic Recording on Thermoplastic Films", *Applied Optics 13*, April 1974, p. 888.

Mathias, J.S., and G.A. Fedde, "Plated-Wire Technology: A Critical Review", *IEEE Transactions on Magnetics*, Vol. MAG-5, No. 4, December 1969, pp. 728-751.

McCallister, J.P., "Plated-Wire Memories: How Far Can We Go?", *IEEE Transactions on Magnetics*, Vol. MAG-6, No. 3, September 1970, pp. 525-528.

Michaelis, P.C., "Magnetic Bubble Mass Memory", *International Magnetics Conference in Toronto*, May 1974.

Micheron, F., C. Mayeux, and J.C. Trotier, "Electrical Control in Photoferroelectric Materials for Optical Storage", *Applied Optics 13*, April 1974, p. 784.

Randell, B., and C.J. Kuehner, "Dynamic Storage Allocation Systems", *Comm. of ACM*, Vol. 11, No. 5, 1968.

Roberts, H.N., J.W. Watkins, and R.H. Johnson, "High Speed Holographic Digital Recorder", *Applied Optics 13*, April 1974, p. 841.

Smith, A.W., "Injection Laser Writing in Chalcogenide Films", *Applied Optics 13*, April 1974, p. 795.

Spain, R.J., H.I. Jauvtis, and F.T. Duben, "DOT Memory Systems", *AFIPS Conference Proceedings 43*, 1974, p. 841.

Sapin, R., and M. Marino, "Magnetic Film Domain-Wall Motion Devices", *IEEE Transactions on Magnetics*, Vol. MAG-6, No. 3, September 1970, pp. 451-463.

Staebler, D.L., and W. Phillips, "Fe-Doped $LiNbO_3$ for Read-Write Applications", *Applied Optics 13*, April 1974, p. 788.

Thaxter, J.B., and M. Kestigian, "Unique Properties of SBN and Their Use in a Layered Optical Memory", *Applied Optics 13*, April 1974, p. 913.

Wolman, E., "A Fixed Optimum Cell-Size for Records of Various Lengths", *Journal of the Association for Computing Machinery*, Vol. 12, No. 1, pp. 53-70.

"Kodak Reports a Cobalt-Iron Crystal as a Medium for Magneto-Optic Storage", *Laser Focus 10*, June 1974, p. 16.

"A Healthier Memory with Organic Materials", *Optical Spectra 8*, June 1974, p. 12.

TECHNICAL SPECIFICATIONS

"Air Force Command and Control Information Processing in the 1980's: Trends in Hardware Technology", *Rand Corporation*.

"Electronic Voice Switching System (EVS)", *Department of Transportation*, Federal Aviation Administration Electronic Equipment, General Requirements, FAA-ER-650-021, 16 August 1971.

"Electronic Voice Switching System", *Department of Transportation*, Federal Aviation Administration Specification, FAA-E-2479, Amendment 2, 22 February 1972.

"Intrasite Communications Equipment Design Specification, PABX-PAR, Design Specification No. 71110005."

"Intrasite Communications Equipment Design Specification, PABX-MSR, Design Specification No. 71210005."

"Intrasite Communications Equipment Design Specification, PABX-BMDC, Design Specification No. 71410005."

"Mission Analysis on Air Force Base Communications - 1985".

"Overseas AUTOVON Network Switching Plan", *DCA Circular*, 370-V185-7.

"Performance Specification, Central Office, Communications, Automatic, AN/TTC-39( )(V)", *TRI-TAC Office*, Fort Monmouth, New Jersey, Specification No. TT-B1-1101-0001, 21 January 1972.

"Specification for Integrated Communications Control System", *Department of Transport*, Telecommunications and Electronics Branch; Design and Construction Division, TED-395, March 1974.

"System Interface Criteria", *DCA Circular*, 370-V175-6.

INTEGRATED CIRCUIT TECHNOLOGY

Altman, Laurence, "Technical Update Solid State, *Electronics 47*, #21, 17 October 1974, p. 138.

Altman, Laurence, "Logic", *Electronics*, 21 February 1974, p. 81-96.

Altman, Laurence, "Technical Update Solid State", *Electronics*, 17 October 1974, pp. 139-151.

Athunus, T.G., "Development of COS/MOS Technology", *Solid State Technology*, June 1974.

Hart, C.M., A. Slob, and H.E.J. Wulmes, "Bipolar LSI Takes a New Direction with Integrated Injection Logic", *Electronics*, 3 October 1974.

Hittinger, W.C., "Metal Oxide Semiconductor Technology", *Scientific American*, August 1973, p. 48.

"Integrated Circuits Bipolar & MOS", *Lecture Notes Infoscope Inc.*, November 1973.

Kaiser, H.W., "CMOS/SOS for High Speed Signal Processing", *IEEE Intercon Technical Paper*, 26-30 March 1973.

Lapidus, G., "Technology '74 Circuits/Devices", *IEEE Spectrum*, June 1974, p. 54.

Powell, M.W., and J.B. Price, "Design and Performance of MOSFET Circuits Using Silicon on Insulating Substrates", *IEEE Intercon Technical Paper*, 26-30 March 1973.

Puckett, H.E., and W.W. Lattin, "Why N-Channel MOS", *IEEE Intercon Technical Paper*, 26-30 March 1973.

Seeds, R.B., and R.L. Luce, "High Density LSI with Isoplanar MOS", *IEEE Intercon Technical Paper*, 26-30 March 1973.

Wolf, H., "The 4-R RAM is On Schedule", *Electronics*, 19 September 1974, p. 76.

SOFTWARE

Baker, F.T., "System Quality Through Structured Programming", *Proceedings FJCC*, 1972, *AFIPS Press*, 1972.

Dahl, O.J., E.W.Dijkstra, and C.A.R. Hoare, "Structured Programming", *New York Academic Press*, 1972.

Dijkstra, E.W., "Structured Programming", *Brussels, Belgium: NATO*, 1970.

Martin, J.J., "Generalized Structured Programming", *Proceedings NCC* 1974, *Montvale, N.J. AFIPS Press*, May 1974.

Maynard, J., "Modular Programming", *New York: Mason and Lipscomb*, 1972.

McGowan, C.L., and J.R. Kelly, "Top Down Structures Programming", *New York: Mason and Lipscomb*, 1974.

Myers, G.J., "Reliable Software Through Composite Design", *New York: Mason and Lipscomb*, 1974.

Parnas, D.L., "On the Criteria to be Used In Decomposing Systems Into Modules", *Comm. ACM*, Vol. 15, No. 12, December 1972.

Lanzano, B.C., "Loader Standardization for Overlay Programs", *Comm. of ACM*, Vol. 12, No. 10, 1969.

Stevens, W.P., G.J. Myers, and L.L. Constantine, "Structured Design", *IBM Systems Journal*, Vol. 13, No. 12, 1974.

Wirth, N., "Systematic Programming", *Prentice-Hall*, Englewood Cliffs, N.J., 1973.

APPENDIX V

TABLE OF CONTENTS

APPENDIX V

TABLE OF CONTENTS - Continued

APPENDIX V

TABLE OF CONTENTS - Continued

APPENDIX V

CIRCUIT SWITCH AND MESSAGE SWITCH FUNCTIONAL BREAKDOWNS AND CALCULATIONS

1.0 C/S SIZES AND TRAFFIC LEVELS

1.1 System Definitions

The purpose of this section is to equate C/S functions which must be performed per second and per Busy Hour (BH) with system sizes and traffic levels. The range of system sizes stipulated in the Statement of Work is from 600 to 6000 terminations. This is taken to mean 600 to 6000 usable outside terminations which are available for lines or trunks. Overhead terminations, those matrix terminations required for register receiver/sender, tone or code busses, operator access, inter-matrix units, conference bridge units or loop-around devices, are in addition to the outside terminations.

Four system sizes are considered. The smallest size being 600 terminations, the largest being 6000 terminations with 2400 and 4200 termination systems as intermediate sizes. This, in effect, considers the effects of adding increments of 1800 terminations. The division of the range of systems into four sizes is entirely arbitrary but is considered adequate to assess the effects of size on the throughput capability requirements of the CPS.

For purposes of these calculations, the mature system traffic levels of Appendix XIII of the AN/TTC-39 specification, TT-B1-1101-0001, 10 May 1973 (BR-128) are used. These traffic levels and call distributions are anticipated for military tactical systems in use in the 1985 time frame.

The calculations are predicated on the assumptions listed below which define particular circuit switch configurations. These are "worst case" configurations from the standpoint of total matrix terminations required and total traffic throughput.

Assumptions

(1) Holding times for voice, data, and TTY calls are as follows:

VOICE CALL	=	240 sec.
DATA CALL	=	32 sec.
TTY CALLS	=	25 sec.

These are average BH holding times.

(2) The distribution of call types is:

VOICE	=	50%
DATA	=	35%

TTY	=	15%

which results in an average holding time of 134.95 sec. The calculations will assume an average holding time of 135 sec.

(3) The percent of loop originated calls assumed to be local-to-local is as follows:

600 Termination Switch	50%
2500 Termination Switch	60%
4200 Termination Switch	70%
6000 Termination Switch	80%

(4) The percent of trunk originated calls assumed to be trunk-to-local is as follows:

600 Termination Switch	50%
2400 Termination Switch	60%
4200 Termination Switch	70%
6000 Termination Switch	80%

(5) Digit collection devices, such as register receiver/sender units or the equivalent digital comma-free code detectors, have the following holding times, which include 2 seconds for "off-hook" to "start of dial":

<u>Analog Register Receiver/Senders</u>

Local-to-Local (incoming)	10 sec.
Local-to-Trunk (incoming)	10 sec.
Local-to-Trunk (outgoing)	1 sec.
Trunk-to-Local (incoming)	1 sec.
Trunk-to-Trunk (incoming)	1 sec.
Trunk-to-Trunk (outgoing)	1 sec.

Where digital code detector-senders are used, common channel signaling is assumed to exist, as in an AN/TTC-39 network. Therefore, no devices are required for local-to-trunk outgoing signaling nor for trunk-to-trunk incoming or outgoing signaling.

<u>Digital Comma-Free Code Detector/Senders</u>

Local-to-Local (incoming)	24 sec.
Local-to-Local (outgoing)	15 sec.
Local-to-Trunk (incoming)	7 sec.

Trunk-to-Local (incoming)	15 sec.
Trunk-to-Local (outgoing)	15 sec.

The above holding times for the digital "registers" assume that in the digital systems, the final COMSEC configuration is employed. The long holding times are required to achieve COMSEC equipment synchronization. The digital "registers" perform COMSEC supervision in addition to the digit collection function.

(6) Where mixed systems are considered, i.e., digital and analog transmission within the same Circuit Switch, all trunk signaling is via common channels and the intermatrix traffic is limited to voice traffic (30% of the total voice traffic).

(7) Loop-around devices are used in all systems with analog transmission matrices. It is assumed that loop-around devices are used on local-to-local and local-to-trunk calls only to achieve signal compatibility. A maximum of 5% of these calls will require the use of loop-around devices.

(8) The following table defines the traffic levels for each switch size. These levels are somewhat higher for the 2400, 4200, and 6000 termination systems than could be extrapolated from the AN/TTC-39 specification, Appendix XIII, but they do reflect possible real world conditions. All values refer to Busy Hour (BH) traffic.

TERMINALS/ SWITCH	CALLS/ TERM.	ORIG. CALLS	ORIG. ERLANGS	LOOP ORIG.	TRUNK ORIG.
600	8	4,800	180E	2,160	2,640
2,400	7	16,800	630E	6,720	10,080
4,200	6	25,200	945E	8,820	16,380
6,000	5	30,000	1,125E	9,000	21,000

Based on the above assumptions, the following calculations will indicate for each of the switch sizes stipulated:

(1) Total calls per switch according to origination and destination, i.e., local or trunk

(2) Total voice calls

(3) Total non-voice (data and TTY)

(4) Total register traffic and number of registers

(5) Total loop-around traffic and number of devices

(6) Total number of matrix terminations required

These results will be reduced, in turn, to the number of C/S functions which must be performed per Busy Hour (BH) and per second (per BH).

(9) All traffic is calculated in Erlangs according to the equation:

$$\text{Erlangs} = \frac{\text{(\#calls/BH)(average Holding time(sec)/call)}}{\text{seconds/BH}}$$

(10) The terminations required for overhead devices such as loop-around devices were derived from full availability trunk loading capacity tables using the Poisson Equation for a grade of service of 1 in 1000.

1.1.1 <u>600 Termination C/S</u>

Originating Erlangs per C/S	=	180
Total Originating Calls	=	4,800
Total Originating Loop Calls	=	2,180
Total Originating Trunk Calls	=	2,640

<u>Loop Calls (50% Local-to-Local)</u>

Local-to-Local	=	1,080
Local-to-Trunk	=	1,080
Local-to-Local (voice)	=	540
Local-to-Trunk (voice)	=	540
Local-to-Local (non-voice)	=	540
Local-to-Trunk (non-voice)	=	540

<u>Trunk Calls (50% Trunk-to-Local)</u>

Trunk-to-Local	=	1,320
Trunk-to-Trunk	=	1,320
Trunk-to-Local (voice)	=	660
Trunk-to-Trunk (voice)	=	660
Trunk-to-Local (non-voice)	=	660
Trunk-to-Trunk (non-voice)	=	660

Originating Voice Calls

Local-to-Local	=	540
Local-to-Trunk	=	540
Trunk-to-Local	=	660
Trunk-to-Trunk	=	660
TOTAL VOICE CALLS	=	2,400

Originating Non-Voice Calls

Local-to-Local	=	540
Local-to-Trunk	=	540
Trunk-to-Local	=	660
Trunk-to-Trunk	=	660
TOTAL NON-VOICE CALLS	=	2,400
TOTAL DATA CALLS	=	1,680
TOTAL TTY CALLS	=	720

Loop-Around Traffic

Local-to-Local Calls	=	1,080
Local-to-Trunk Calls	=	1,080
TOTAL CALLS	=	2,160
x5%	=	108
Loop-Around Traffic	=	4.05E
Number of Loop-Around Devices	=	12

Register Traffic (Analog System)

Local-to-Local (incoming)	=	3E
Local-to-Trunk (incoming)	=	3E
Local-to-Trunk (outgoing)	=	.3E
Trunk-to-Local (incoming)	=	.37E
Trunk-to-Trunk (incoming)	=	.37E
Trunk-to-Trunk (outgoing)	=	.37E
TOTAL REGISTER TRAFFIC	=	7.41E
TOTAL REGISTERS	=	12

Register Traffic (Digital System)

Local-to-Local (incoming)	=	7.2E

Local-to-Local (outgoing)	=	4.5E
Local-to-Trunk (incoming)	=	2.1E
Trunk-to-Local (incoming)	=	5.5E
Trunk-to-Local (outgoing)	=	5.5E
TOTAL REGISTER TRAFFIC	=	24.8E
TOTAL DIGITAL REGISTERS	=	32

Total Terminations (Analog System)

Outside Terminations	=	600
Loop-Around (2 ea/device)	=	24
Register	=	12
TOTAL TERMINATIONS	=	636

Total Terminations (Digital System)

Outside Terminations	=	600
LKG (2 ea/device)	=	64
Register (NOTE: 1 LKG required/register)	=	32
TOTAL TERMINATIONS	=	696

The calculations for the 2400, 4200, and 6000 termination C/S sizes were performed as shown for the 600 termination switch. Listed below are the results of these calculations.

1.1.2 2400 Termination C/S

Originating Erlangs per C/S	=	630E
Total Originating Calls	=	16,800
Total Originating Loop Calls	=	6,720
Total Originating Trunk Calls	=	10,080

Loop Calls (60% Local-to-Local)

Local-to-Local	=	4,032
Local-to-Trunk	=	2,688

Trunk Calls (60% Trunk-to-Local)

Trunk-to-Local	=	6,048
Trunk-to-Trunk	=	4,032

Originating Voice Calls	=	8,400
Originating Non-Voice Calls	=	8,400
Data Calls	=	5,880
TTY Calls	=	2,520
Loop-Around Traffic		
Local-to-Local Calls	=	4,032
Local-to-Trunk Calls	=	2,688
TOTAL	=	6,720
x5%	=	336
Traffic	=	12.6E
Loop-Around Devices	=	26
Register Traffic (Analog System)		
TOTAL REGISTER TRAFFIC	=	23.325E
Registers required	=	29
Register Traffic (Digital System)		
Total Register Traffic	=	99.3E
Registers required	=	110
Total Terminations (Analog System)		
Outside	=	2,400
LKG (2 ea/device)	=	220
Register	=	110
TOTAL	=	2,730

1.1.3 4200 Termination C/S

Originating Erlangs per C/S	=	945E
Total Originating Calls	=	25,200
Total Originating Loop Calls	=	8,820
Total Originating Trunk Calls	=	16,380
Loop Calls (70% Local-to-Local)		
Local-to-Local	=	6,174
Local-to-Trunk	=	2,645

Trunk Calls (70% Trunk-to-Local)

Trunk-to-Local	= 11,466
Trunk-to-Trunk	= 4,914

Originating Voice Calls = 12,600

Originating Non-Voice = 12,600

Data	= 8,820
TTY	= 3,780

Loop-Around Traffic

Loop Originated Calls	= 8,820
x5%	= 441
Traffic	= 16.537E
Loop-Around Devices	= 32

Register Traffic (Analog System)

Total Register Traffic	= 31.147E
Registers required	= 37

Register Traffic (Digital System)

Total Register Traffic	=167.578E
Registers required	= 170

Total Terminations (Analog System)

Outside	= 4,200
Loop-Around (2 ea/device)	= 64
Register	= 37
TOTAL	= 4,301

Total Terminations (Digital System)

Outside	= 4,200
LKG (2 ea/device)	= 340
Register	= 170
TOTAL	= 4,710

1.1.4 <u>6000 Termination C/S</u>

Originating Erlangs per C/S	= 1,125E
Total Originating Calls	= 30,000
Total Originating Loop Calls	= 9,000
Total Originating Trunk Calls	= 21,000
<u>Loop Calls (80% Local-to-Local)</u>	
Local-to-Local	= 7,200
Local-to-Trunk	= 1,800
<u>Trunk Calls (80% Trunk-to-Local)</u>	
Trunk-to-Local	= 16,800
Trunk-to-Trunk	= 4,200
<u>Originating Voice Calls</u>	= 15,000
<u>Originating Non-Voice Calls</u>	= 15,000
Data Calls	= 10,500
TTY Calls	= 4,500
<u>Loop-Around Traffic</u>	
Loop Originated Calls	= 9,000
x5%	= 450
Traffic	= 16.875E
Loop-Around Devices	= 32
<u>Register Traffic (Analog System)</u>	
Total Register Traffic	= 33.48E
Registers required	= 39
<u>Register Traffic (Digital System)</u>	
Total Register Traffic	= 221.5E
Registers required	= 225
<u>Total Terminations (Analog System)</u>	
Outside	= 6,000
Loop-Around (2 ea/device)	= 64

Register	= 39
TOTAL	= 6,103

<u>Total Terminations (Digital System)</u>

Outside	= 6,000
LKG (2 ea/device)	= 450
Register	= 225
TOTAL	= 6,675

1.2 Breakdown of C/S Functions

This paragraph describes in terms of call types each of the C/S functions which must be performed.

Certain assumptions are made concerning the system configuration which are:

(1) Digit collection and sending devices (ASF units) are pooled in a single group.

(2) Pre-emption and conferencing features are not exercised.

(3) Attendant (or Operator) positions are equipped with direct data paths to the control system and do not require register connections for "call forwarding".

(4) The LKG's (TENLEY Equipment) are pooled in a single group.

(5) The loop-around devices required for call compatibility are pooled in a single group.

(6) Information tones:

  (a) Dial tone is provided by the ASF units.

  (b) Half connection tones, tones which are sent to a subscriber when a connection will not exist (such as busy tone), are sent by the SSF.

  (c) Split connection tones, tones which are sent to a subscriber when a connection will exist (such as ring, ringback, etc.) are sent by the MCF.

(7) All subscriber voice end instruments use either AC supervision signaling and DTMF address signaling or comma-free codes for both.

(8) All data or TTY subscriber terminals terminated on the C/S use dial-up facilities similar to the voice end instruments to signal the C/S.

(9) All trunk signaling is via common channel messages.

1.2.1 Matrix Control Function

The matrix control operations considered are the three basic functions, Connect A to B, Disconnect A, and Inject Tone (or code). These are the minimum operations and do not include path pre-emptions nor diagnostic operations. The basic functions can be further subdivided as follows:

Connect A to B

(1) Analyze command

(2) Find A (determine if Terminal A is idle or busy)

(3) Find B (determine if Terminal B is idle or busy)

(4) Search for path between A and B

(5) Store path

(6) Connect path

(7) Trace path from A to B

(8) Trace path from B to A

(9) Format and send "A connected to B" message to CPF

Disconnect A

(1) Analyze command

(2) Trace A (trace path from A to B)

(3) Store path

(4) Disconnect path

(5) Find A (insure that A is idle)

(6) Find B (insure that B is idle)

(7) Format and send "A disconnected from B" message to CPF

Inject Tone/Code to A - (NOTE: The same command is used to stop tones or codes).

(1) Analyze command

(2) Find A

(3) Inject Tone or code (or "no tone" or "no code")

(4) Format "Tone Injected to A" message to CPF

NOTE: If a Connect A to B command is issued and B is already connected to another terminal, i.e., B is busy, then the connect sequence is as follows:

Connect A to B

(1) Analyze command

(2) Find A

(3) Find B

(4) Format and send "B is busy" message to CPF

1.2.1.1 Matrix Operations (Analog System)

Local-to-Local Calls

(1) Connect calling party to ASF unit

(2) Disconnect from ASF unit

(3) Connect to called party's termination

(4) Send ring to called party

(5) Send ringback to calling party

(6) Stop ring signal

(7) Stop ringback signal

(8) Disconnect called from calling

Local-to-Trunk Calls

(1) Connect calling party to ASF unit

(2) Disconnect from ASF unit

(3) Connect calling party to trunk

(4) Disconnect call

Trunk-to-Local Calls

(1) Connect trunk to called party's termination

(2) Send ring to called party

(3) Send ringback to trunk

(4) Stop ring

(5) Stop ringback

(6) Disconnect call

Trunk-to-Trunk Calls

(1) Connect calling trunk to called trunk

(2) Disconnect calling trunk from called trunk

Loop-Around Calls (same as Local-to-Local and Local-to-Trunk with addition of the following:)

(1) Connect called party to loop-around device

(2) Connect calling party to loop-around device

(3) Disconnect called party from device

(4) Disconnect calling party from device

Operator Forwarded Calls (1.0% of the total Local-to-Local and Local-to-Trunk calls) are the same as the above sequences with the following added steps:

(1) Connect calling party to operator

(2) Connect called party to operator

(3) Disconnect calling party from operator

(4) Disconnect called party from operator

A Call Busy Factor (CBF) of 25% is recommended by the AN/TTC-39 specification, Appendix XIII. (BR-128)

Call Busy Factor Adder

Local-to-Local

(1) Connect calling party to ASF unit

(2) Disconnect calling party from ASF unit

(3) Connect to calling to called (return "B is Busy" message)

Local-to-Trunk

(1) Connect A to ASF unit

(2) Disconnect A from ASF unit

Trunk-to-Local

(1) Connect trunk to called terminal (returns "B is Busy" message)

1.2.1.2 Matrix Operations (Digital System)

Local-to-Local

(1) Connect calling party to LKG #1

(2) Connect LKG #1 to ASF unit

(3) Connect called party to LKG #2

(4) Connect LKG #2 to ASF unit

(5) Disconnect LKG #1 from ASF unit

(6) Disconnect LKG #2 from ASF unit

(7) Connect LKG #1 to LKG #2

(8) Send ring signal to called

(9) Send ringback to calling

(10) Stop ring signal

(11) Stop ringback signal

(12) Disconnect LKG #1 from LKG #2

(13) Disconnect calling from LKG #1

(14) Disconnect called from LKG #2

(15) Connect calling to called

(16) Disconnect calling from called

Local-to-Trunk

(1) Connect calling party to ASF unit

(2) Disconnect from ASF unit

(3) Connect calling party to trunk

(4) Disconnect calling party from trunk

Trunk-to-Local

(1) Connect trunk to LKG #1

(2) Connect LKG #1 to ASF unit

(3) Connect called party to LKG #2

(4) Connect LKG #2 to ASF unit

(5) Disconnect LKG #1 from ASF unit

(6) Disconnect LKG #2 from ASF unit

(7) Connect LKG #1 to LKG #2

(8) Send ring to called party

(9) Send ringback to trunk

(10) Stop ring signal

(11) Stop ringback signal

(12) Disconnect LKG #1 from LKG #2

(13) Disconnect trunk from LKG #1

(14) Disconnect called party from LKG #2

(15) Connect trunk to called party

(16) Disconnect trunk from called party

<u>Trunk-to-Trunk</u>

(1) Connect calling trunk to called trunk

(2) Disconnect calling trunk from called trunk

Operator forwarded calls add the same number of matrix operations as in an analog matrix.

<u>CBF Adders</u>

<u>Local-to-Local</u>

(1) Connect calling party to LKG #1

(2) Connect LKG #1 to ASF unit

(3) Connect called to LKG #2 (returns "B is Busy" message)

(4) Disconnect LKG #1 from ASF unit

(5) Disconnect LKG #1 from calling party

<u>Local-to-Trunk</u>

(1) Connect calling party to ASF unit

(2) Disconnect from ASF unit

(3) Connect calling party to trunk

(4) Disconnect calling party from trunk

<u>Trunk-to-Local</u>

(1) Connect trunk to LKG #1

(2) Connect LKG #1 to ASF unit

(3) Connect called to LKG #2 (returns "B is Busy" message)

(4) Disconnect trunk from LKG #1

(5) Disconnect LKG #1 from ASF unit

Trunk-to-Trunk

(1) Connect calling trunk to called trunk

(2) Disconnect trunk

1.2.1.3 Summary of Matrix Control Operations

Analog System

Call Type	Matrix Operations
Local-to-Local	8
Local-to-Trunk	4
Trunk-to-Local	6
Trunk-to-Trunk	2
Loop-Around Adder	
Local-to-Local	4
Local-to-Trunk	4
Operator Forwarded (Adder)	
Local-to-Local	4
Local-to-Trunk	4
Call Busy Factor (Adder)	
Local-to-Local	3
Local-to-Trunk	2
Trunk-to-Local	1
Trunk-to-Trunk	0
Digital System	
Local-to-Local	16
Local-to-Trunk	4
Trunk-to-Local	16
Trunk-to-Trunk	2
Operator Forwarded (Adder)	
Local-to-Local	4
Local-to-Trunk	4

	Matrix Operations
CBF (Adder)	
Local-to-Local	5
Local-to-Trunk	4
Trunk-to-Local	5
Trunk-to-Trunk	2

1.2.2 Supervision Signaling (SSF) Functional Operations

The purpose of this section is to describe in further detail the operations required of a C/S to accomplish the functions described in paragraph 2.2.2.3.

1.2.2.1 SSF - Fixed Operations

The supervision functional operations can be divided into two main types. The first type includes all functional operations which are independent of the traffic levels. These include: (1) the scan, (2) the compare, and (3) the change of state detection functional operations.

The scan consists of sampling each terminal in a system at a sufficient rate so that all required signal timing can be accurately done. From the discussion in paragraph 2.2.2.3, this requires that each terminal be examined every ten milliseconds. The operation requires that the system be able to multiplex a large number of data points into one register or comparator circuit. The scan must be run from a real time clock so that the samples are taken at a fixed rate.

The compare functional operation consists of a logical compare of a present sample of the status of a terminal with the previous status. This requires data to be addressed and recalled from memory and then to be compared with the input from the scanned sample. As a result of the comparison, a flag must be set if a difference exists.

The detection of change of state operations uses the flag set by the compare operation to determine when an update of status information and notification to the signal validation operation must occur. Like the scan and compare functional operations, the detection operation is fixed for each size of switch and only the details of the operation as a result of a detected change of state is dependent on the call rate for the system.

1.2.2.2 SSF Operations - Call Related

Analog

Local-to-Local

(1) Detects off-hook signal - terminal A

(2) Stores new terminal A status

(3) Analyzes classmark of requesting terminal - determines signaling convention

(4) Validates signal - minimum time-out

(5) Formats message to CPF - service request for specific terminal

(6) Detects off-hook signal - terminal B

(7) Stores new terminal B status

(8) Analyzes classmark of requesting terminal - determines signaling convention

(9) Validates signal - minimum time-out

(10) Formats message to CPF - terminal off-hook

(11) Detects on-hook signal - A or B terminal

(12) Stores new terminal status

(13) Analyzes classmark of terminal - determines signaling convention

(14) Validates signal - minimum time-out

(15) Formats message to CPF - terminal A or B on-hook

(16) Receives message from CPF - send "busy" to terminal B or A

(17) Sends "busy" tone to terminal

(18) Detects on-hook signal

(19) Stores new terminal status

(20) Analyzes classmark of terminal

(21) Validates signal

(22) Formats message to CPF - terminal on-hook

(23) Receives message from CPF - stop busy tone to terminal

(24) Stops busy tone to terminal

Local-to-Trunk

(1) Detects off-hook signal from terminal A

(2) Stores new status for terminal A

(3) Analyzes classmark of terminal A - determines signaling convention

(4) Validates signal

(5) Formats message to CPF - terminal A - off-hook

(6) Receives "call initiate" common channel message from CPF

(7) Sends "call initiate" via common channel to another switch

(8) Receives common channel message - acknowledgement

(9) Formats common channel acknowledgement message to CPF

(10) Receives "call complete" message on common channel

(11) Sends "acknowledge" on common channel

(12) Formats "call complete" message to CPF

(13) Detects on-hook from terminal

(14) Stores new status for terminal

(15) Analyzes classmark for terminal

(16) Validates signal

(17) Formats terminal "on-hook" message to CPF

(18) Receives common channel "release" message from CPF

(19) Sends "release" message via common channel

(20) Receives "acknowledge" on common channel

(21) Formats release acknowledge message to CPF

<u>Trunk-to-Local</u>

(1) Receives "call initiate" common channel message

(2) Formats "call initiate" message to CPF

(3) Receives "acknowledge" message from CPF

(4) Sends "acknowledge" message via common channel

(5) Receives "call complete" message from CPF

(6) Formats and sends "call complete" via common channel

(7) Receives "acknowledgement" from common channel

(8) Detects terminal off-hook

(9) Stores new terminal status

(10) Analyzes classmark for terminal

(11) Validates signal

(12) Formats message "terminal off-hook" to CPF

(13) Receives "release" message from common channel

(14) Formats and sends "release" message to CPF

(15) Receives "acknowledge" message from CPF

(16) Formats and sends "acknowledge" via common channel

(17) Receives message from CPF - send terminal A "busy"

(18) Sends "busy" tone to terminal A

(19) Detects terminal A on-hook

(20) Stores new terminal status

(21) Analyzes classmark for terminal

(22) Validates signal

(23) Send terminal A on-hook message to CPF

(24) Receives message from CPF - stop busy to terminal A

(25) Stops busy tone to terminal A

Trunk-to-Trunk

(1) Receives "call initiate" from common channel

(2) Sends "call initiate" message to CPF

(3) Receives "acknowledge" from CPF

(4) Sends "acknowledge" via common channel

(5) Receives "call initiate" message from CPF

(6) Sends "call initiate" message via common channel

(7) Receives "acknowledge" from common channel

(8) Formats "acknowledge" message to CPF

(9) Receives "call complete" message from common channel

(10) Formats "call complete" message to CPF

(11) Receives "acknowledge" message from CPF

(12) Sends "acknowledge" message via common channel

(13) Receives "call complete" message from CPF

(14) Sends "call complete" message via common channel

(15) Receives "acknowledge" message from common channel

(16) Sends "acknowledge" message to CPF

(17) Receives "release" message from common channel

(18) Sends "release" message to CPF

(19) Receives "acknowledge" message from CPF

(20) Sends "acknowledge" message via common channel

(21) Receives "release" message from CPF

(22) Sends "release" message via common channel

(23) Receives "acknowledge" message from common channel

(24) Sends "acknowledge" message to CPF

Call Busy Factor (CBF) Adder

Local-to-Local

(1) Detects off-hook signal - terminal A

(2) Stores new terminal A status

(3) Analyzes terminal classmark

(4) Validates signal

(5) Sends terminal A - off-hook message to CPF

(6) Receives message from CPF - send busy to A

(7) Sends "busy" signal to terminal A

(8) Detects terminal A goes on-hook signal

(9) Stores new terminal status

(10) Analyzes classmark for terminal

(11) Validates on-hook signal

(12) Sends terminal A "on-hook" message to CPF

(13) Receives "stop busy tone" message from CPF

(14) Stop busy tone to terminal A

(15) Updates terminal status

Local-to-Trunk

(1) Detects terminal A change of state

(2) Store new terminal A status

(3) Analyzes terminal classmark

(4) Validates off-hook signal

(5) Sends terminal A - off-hook message to CPF

(6) Receives "call initiate" message from CPF

(7) Sends "call initiate" message via common channel

(8) Receives "acknowledge" from common channel

(9) Sends "acknowledge" to CPF

(10) Receives "busy" message from common channel

(11) Sends "busy" message to CPF

(12) Receives "acknowledge" message from CPF

(13) Sends "acknowledge" message via common channel

(14) Receives message from CPF - send busy tone to A

(15) Sends "busy" tone to terminal A

(16) Detects change of state for terminal A

(17) Stores new terminal A status

(18) Analyzes classmark for terminal A

(19) Validates on-hook signal for terminal A

(20) Sends terminal A on-hook message to CPF

(21) Receives message from CPF - stop busy tone to A

(22) Stops busy tone to terminal A

<u>Trunk-to-Local</u>

(1) Receives "call initiate" message on common channel

(2) Sends "call initiate" message to CPF

(3) Receives "acknowledge" message from CPF

(4) Sends "acknowledge" message via common channel

(5) Receives "busy terminal" message from CPF

(6) Sends "busy terminal" message via common channel

(7) Receives "acknowledge" on common channel

(8) Sends "acknowledge" to CPF

<u>Trunk-to-Trunk</u>

(1) Receives "call initiate" message on common channel

(2) Sends "call initiate" message to CPF

(3) Receives "acknowledge" message from CPF

(4) Sends "acknowledge" message via common channel

(5) Receives "call initiate" message from CPF

(6) Sends "call initiate" message via common channel

(7) Receives "acknowledge" message on common channel

(8) Sends "acknowledge" message to CPF

(9) Receives "busy" message on common channel

(10) Sends "busy" message to CPF

(11) Receives "acknowledge" from CPF

(12) Sends "acknowledge" via common channel

(13) Receives "busy" message from CPF

(14) Sends "busy" message via common channel

(15) Receives "acknowledge" message on common channel

(16) Sends "acknowledge" message to CPF

(17) Receives "release" message from CPF

(18) Sends "release" message via common channel

(19) Receives "acknowledge" message on common channel

(20) Sends "acknowledge" message to CPF

1.2.2.3 SSF Operations - Digital

<u>Local-to-Local</u>

(1) Detects "request for service" for terminal A

(2) Sends "request for service" message to CPF

(3) Detects "release" from terminal A or B

(4) Sends "release" message to CPF

(5) Receives message from CPF - send "busy" to terminal X

(6) Sends "busy" to terminal X

(7) Detects "release" from terminal X

(8) Sends "release" message to CPF

(9) Receives message from CPF - stop "busy"

(10) Stops "busy" to terminal X

<u>Local-to-Trunk</u>

(1) Detects A - off-hook

(2) Sends terminal A off-hook message to CPF

(3) Receives "call initiate" message from CPF

(4) Sends "call initiate" message via common channel

(5) Receives "acknowledge" on common channel

(6) Sends "acknowledge" message to CPF

(7) Receives "call complete" message on common channel

(8) Sends "call complete" message to CPF

(9) Receives "acknowledge" message from CPF

(10) Sends "acknowledge" message via common channel

(11) Detects A on-hook

(12) Sends "A on-hook" message to CPF

(13) Receives "release" message from CPF

(14) Sends "release" message via common channel

(15) Receives "acknowledge" message on common channel

(16) Sends "acknowledge" message to CPF

<u>Trunk-to-Local</u>

(1) Receives "call initiate" message on common channel

(2) Sends "call initiate" message to CPF

(3) Receives "acknowledge" message from CPF

(4) Sends "acknowledge" message via common channel

(5) Receives "call complete" message from CPF

(6) Sends "call complete" message via common channel

(7) Receives "acknowledge" message on common channel

(8) Sends "acknowledge" message to CPF

(9) Receives "release" message on common channel

(10) Sends "release" message to CPF

(11) Receives "acknowledge" message from CPF

(12) Sends "acknowledge" via common channel

(13) Receives message from CPF - send "busy" to terminal A

(14) Sends "busy" to terminal A

(15) Receives "release" from terminal A

(16) Sends "release" to CPF

(17) Receives message from CPF - stop "busy"

(18) Stops "busy" to terminal A

<u>Trunk-to-Trunk</u>

(1) Receives "call initiate" message on common channel

(2) Sends "call initiate" message to CPF

(3) Receives "acknowledge" message from CPF

(4) Sends 'acknowledge" message via common channel

(5) Receives "call initiate" message from CPF

(6) Sends "call initiate" message via common channel

(7) Receives "acknowledge" message on common channel

(8) Sends "acknowledge" message to CPF

(9) Receives "call complete" message on common channel

(10) Sends "call complete" message to CPF

(11) Receives "acknowledge" message from CPF

(12) Sends "acknowledge" message via common channel

(13) Receives "call complete" message from CPF

(14) Sends "call complete" message via common channel

(15) Receives "acknowledge" message on common channel

(16) Sends "acknowledge" message to CPF

(17) Receives "release" message on common channel

(18) Sends "release" message to CPF

(19) Receives "acknowledge" message from CPF

(20) Sends "acknowledge" message via common channel

(21) Receives "release" message from CPF

(22) Sends "release" message via common channel

(23) Receives "acknowledge" message on common channel

(24) Sends "acknowledge" message to CPF

<u>Call Busy Factor Adder</u>

<u>Local-to-Local</u>

(1) Receives "seize" from terminal

(2) Sends "seize" to CPF

(3) Receives message from CPF - send "busy"

(4) Sends "busy" to terminal

(5) Detects "release" from terminal A

(6) Sends "release" message to CPF

(7) Receives message from CPF - stop "busy"

(8) Stops "busy" to terminal A

Local-to-Trunk

(1) Detects "seize" from terminal A

(2) Sends "seize" message to CPF

(3) Receives "call initiate" message from CPF

(4) Sends "call initiate" message via common channel

(5) Receives "acknowledge" message on common channel

(6) Sends "acknowledge" message to CPF

(7) Receives "busy" message on common channel

(8) Sends "busy" message to CPF

(9) Receives "acknowledge" message from CPF

(10) Sends "acknowledge" message via common channel

Trunk-to-Local

(1) Receives "call initiate" message on common channel

(2) Sends "call initiate" message to CPF

(3) Receives "acknowledge" message from CPF

(4) Sends "acknowledge" message via common channel

(5) Receives "busy" message from CPF

(6) Sends "busy" message via common channel

(7) Receives "acknowledge" message on common channel

(8) Sends "acknowledge" message to CPF

Trunk-to-Trunk

(1) Receives "call initiate" message on common channel

(2) Sends "call initiate" message to CPF

(3) Receives "acknowledge" message from CPF

(4) Sends "acknowledge" message via common channel

(5) Receives "call initiate" message from CPF

(6) Sends "call initiate" message via common channel

(7) Receives "acknowledge" message on common channel

(8) Sends "acknowledge" message to CPF

(9) Receives "busy" message on common channel

(10) Sends "busy" message to CPF

(11) Receives "acknowledge" message from CPF

(12) Sends "acknowledge" message via common channel

(13) Receives "busy" message from CPF

(14) Sends "busy" message via common channel

(15) Receives "acknowledge" message on common channel

(16) Sends "acknowledge" message to CPF

(17) Receives "release" message on common channel

(18) Sends "release" message to CPF

(19) Receives "acknowledge" message from CPF

(20) Sends "acknowledge" message via common channel

(21) Receives "release" message from CPF

(22) Sends "release" message via common channel

(23) Receives "acknowledge" message on common channel

(24) Sends "acknowledge" message to CPF

1.2.2.4 Summary of SSF Operations

<u>Fixed Operations</u>

3 operations/terminal

Call Related - Analog	Total Operations
Local-to-Local	24
Local-to-Trunk	21
Trunk-to-Local	25
Trunk-to-Trunk	24
Call Busy Factor Adder	
Local-to-Local	15
Local-to-Trunk	22
Trunk-to-Local	8

	Total Operations
Trunk-to-Trunk	20
Digital	
Local-to-Local	10
Local-to-Trunk	16
Trunk-to-Local	18
Trunk-to-Trunk	24
Call Busy Factor Adder	
Local-to-Local	8
Local-to-Trunk	10
Trunk-to-Local	8
Trunk-to-Trunk	24

1.2.3 ASF Functional Breakdown

The following assumptions are made concerning call types:

(1) Local-to-Local (L-L) calls require 7-digit dialing

(2) Local-to-Trunk (L-T) calls require "1+" dialing or 11-digits

Local-to-Local Calls

(1) Receives message from CPF identifying ASF port number and type of signaling

(2) Sends dial tone (seize acknowledge)

(3) Receives first digit and stores

(4) Removes dial tone

(5) Validates first digit (multiple sample - 3 samples)

(6) Analyzes digit with respect to numbering plan - validates with respect to numbering plan

(7) Sets last digit marker - examination of first digit indicates total number of digits to be received

(8) Detects and times interdigit

(9) Detects second digit - stores digit

(10) Validates digit - multiple samples

(11) Analyzes digit with respect to numbering plan

(12) Repeats steps 8, 9, 10, and 11 for third, fourth, fifth, sixth and seventh digits which adds 20 steps

.
.
.
.

(32) Sends first three digits to CPF (office exchange - allows CPF to start routing procedures)

(33) Send last four digits to CPF - indicates that its task is finished

(34) Receives acknowledgement from CPF - releases ASF port - resets signaling type designator

Local-to-Trunk Calls

Local-to-Trunk calls include all of the Local-to-Local steps plus the following adder:

(1) Sends first three digits after the "1" to CPF (in this case they are the AREA code)

(2) Repeats steps 8, 9, 10, and 11 for the eighth, *ninth*, *tenth*, and *eleventh* digits which adds 16 steps.

(3) Total steps for L-T call equals 51.

Call Busy Factor Adder

(1)	L-L	=	35
(2)	L-T	=	51

1.2.4 CPF Functional Breakdown

The CPF functional breakdown is based on the assumptions listed in paragraph 2.4.2 and assumes traffic metering.

Traffic Metering requirements:

(a) All ASF ports busy

(b) All trunks (in a trunk group) busy

(c) Matrix blocking conditions

(d) Cumulative record of the total number of incoming and outgoing calls offered to each individual trunk group

(e) Cumulative record of the number of calls originated by subscribers within a specific line group assignment

(f) Cumulative record of the number of pre-emptions on each trunk group

(g) Cumulative record of the number of lost calls in each precedence category

(h) Cumulative record of the number of conference calls (and number of conferees) in each precedence category

(i) Cumulative record of the number of pre-emptions on local subscriber lines

(j) Total number of attempts and usage on each common equipment group

(k) Total number of circuits busy

1.2.4.1 CPF Operations (Analog)

<u>Local-to-Local</u>

(1) Receives request for service from SSF

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (Connect A to ASF port)

(4) Receives and analyzes reply from MCF

(5) Formats message to ASF

(6) Formats message to OMF (ASF unit usage report)

(7) Receives and analyzes reply from ASF (receives digits)

(8) Formats message to MCF (disconnect A from ASF port)

(9) Sends message to OMF (ASF usage update)

(10) Receives and analyzes reply from MCF

(11) Analyzes digits, performs digit translation

(12) Analyzes classmark of called terminal

(13) Formats message to MCF (connect A to B)

(14) Receives and analyzes reply from MCF

(15) Formats busy circuit message to OMF

(16) Formats message to MCF (inject ring to B)

(17) Receives and analyzes reply

(18) Formats message to MCF (inject ringback to A)

(19) Receives and analyzes reply

(20) Receives off-hook message from SSF (B goes off-hook)

(21) Formats message to MCF (stop ring to B)

(22) Receives and analyzes reply from MCF

(23) Formats message to MCF (stop ringback to A)

(24) Receives and analyzes reply from MCF

(25) Receives release information from SSF (A or B releases)

(26) Formats message to MCF (disconnect A or B)

(27) Receives and analyzes reply from MCF

(28) Analyzes classmark of remaining terminal

(29) Formats report message to OMF (busy circuit update)

(30) Directs SSF to send busy to remaining party

(31) Receives release information from SSF (remaining party)

(32) Directs SSF to stop busy tone

(33) Updates status table

Local-to-Trunk

(1) Receives request for service from SSF (A goes off-hook)

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (connect A to ASF port)

(4) Receives and analyzes reply from MCF

(5) Formats message to ASF

(6) Formats message to OMF (ASF usage report)

(7) Receives and analyzes reply from ASF (receives digits)

(8) Formats message to MCF (disconnect A from ASF port)

(9) Sends message to OMF (ASF usage update)

(10) Receives and analyzes reply from MCF

(11) Analyzes digits, performs digit translation

(12) Performs routing, determines primary trunk group

(13) Examines trunk status table, selects idle trunk, marks trunk

(14) Analyzes trunk classmark

(15) Formats "call initiate" common channel message, sends to SSF

(16) Receives confirmation from SSF

(17) Formats message to OMF (trunk usage message)

(18) Formats message to MCF (connect A to trunk)

(19) Receives and analyzes reply from MCF

(20) Receives "call complete" message from SSF

(21) Receives message from SSF (A goes on-hook)

(22) Formats message to MCF (disconnect A)

(23) Receives and analyzes reply from MCF

(24) Analyzes classmark of remaining terminal

(25) Formats common channel (release) message, send to SSF

(26) Receives confirmation of trunk release from SSF

(27) Formats trunk status message to OMF

(28) Updates status tables

Trunk-to-Local

(1) Receives "call initiate" message from SSF

(2) Analyzes message, performs routing (digit translation)

(3) Analyzes classmark of called terminal

(4) Sends "acknowledge" message to SSF

(5) Formats message to MCF (connect trunk to A)

(6) Receives and analyzes reply from MCF

(7) Formats "call complete" message to SSF

(8) Formats message to MCF (send ring to A)

(9) Receives and analyzes reply from MCF

(10) Formats message to MCF (send ringback to trunk)

(11) Receives and analyzes reply from MCF

(12) Formats message to OMF (trunk usage)

(13) Receives "release" message from SSF

(14) Sends "acknowledge" message to SSF

(15) Formats message to OMF (status update)

(16) Formats message to MCF (disconnect trunk)

(17) Receives and analyzes reply from MCF

(18) Sends message to SSF (send busy to A)

(19) Receives message from SSF (A goes on-hook)

(20) Sends message to SSF (stop busy to A)

(21) Updates tables

<u>Trunk-to-Trunk</u>

(1) Receives "call initiate" message from SSF

(2) Sends "acknowledge" to SSF

(3) Analyzes message, performs routing

(4) Examines trunk status tables, selects outgoing trunk

(5) Analyzes classmark of trunk

(6) Formats "call initiate" message to SSF

(7) Receives "acknowledgement" from SSF

(8) Formats message to MCF (connect trunk A to trunk B)

(9) Receives and analyzes reply from MCF

(10) Send status message to OMF (trunk usage)

(11) Receives "call complete" message from SSF

(12) Sends "acknowledge" message to SSF

(13) Sends "call complete" message from SSF

(14) Receives "acknowledge" message from SSF

(15) Receives "release" message from SSF

(16) Send "acknowledge" message to SSF

(17) Formats message to MCF (disconnect trunk A)

(18) Receives and analyzes reply from MCF

(19) Sends "release" message to SSF

(20) Receives "acknowledge" message from SSF

(21) Send status message to OMF

(22) Update tables

<u>Loop-Around Adder</u>

(1) Formats message to MCF (connect A to SEF device)

(2) Receives and analyzes reply from MCF

(3) Formats message to MCF (connect B to SEF device)

(4) Receives and analyzes reply from MCF

(5) Sends status message to OMF (SEF equipment usage)

(6) Formats message to MCF (disconnect A from SEF device)

(7) Receives and analyzes reply from MCF

(8) Formats message to MCF (disconnect B from SEF device)

(9) Receives and analyzes reply from MCF

(10) Send status message to OMF

(11) Update tables

NOTE: The loop-around adder is the same for Local-to-Local and Local-to-Trunk calls.

Operator Forwarded Adder

(1) Sends message to OMF (operator service request)

(2) Receives message from OMF (operator assignment)

(3) Formats message to MCF (connect A to OPR)

(4) Receives and analyzes reply

(5) Receives message from OMF (outgoing OPR terminal B)

(6) Analyzes classmark of terminal

(7) Formats message to MCF (connect OPR to B)

(8) Receives and analyzes reply from MCF

(9) Receives message from OMF (drop OPR connection A & B)

(10) Formats message to MCF (disconnect A from OPR)

(11) Receives and analyzes reply

(12) Formats message to MCF (disconnect B from OPR)

(13) Receives and analyzes reply

Call Busy Factor (CBF) Adder

Local-to-Local

(1) Receives request for service from SSF

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (connect A to ASF port)

(4) Receives and analyzes reply from MCF

(5) Formats message to ASF

(6) Formats message to OMF (ASF usage report)

(7) Receives and analyzes reply from ASF (receives digits)

(8) Formats message to MCF (disconnect A from ASF port)

(9) Sends message to OMF (ASF usage report)

(10) Receives and analyzes reply from MCF

(11) Analyzes digits, perform digit translation

(12) Analyzes classmark of called terminal

(13) Formats message to MCF (connect A to B)

(14) Receives and analyzes reply from MCF

(15) Sends message to SSF (send busy to A)

(16) Receives message from SSF (A goes on-hook)

(17) Sends message to SSF (stop busy tone)

(18) Update tables

(19) Sends message to OMF (circuit usage report)

Local-to-Trunk

(1) Receives request for service from SSF (A goes off-hook)

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (connect A to ASF port)

(4) Receives and analyzes reply from MCF

(5) Formats message to ASF

(6) Formats message to OMF (ASF usage report)

(7) Receives and analyzes reply from ASF (receives digits)

(8) Formats message to MCF (disconnect A from ASF port)

(9) Sends message to OMF (ASF usage update)

(10) Receives and analyzes reply from MCF

(11) Analyzes digits, performs digit translation

(12) Performs routing, determines primary trunk group

(13) Examines trunk status table, selects idle trunk, marks trunk)

(14) Analyzes classmark of trunk

(15) Formats "call initiate" common channel message, sends to SSF

(16) Receives confirmation from SSF

(17) Formats message to OMF (trunk usage message)

(18) Formats message to MCF (connect A to trunk)

(19) Receives and analyzes reply from MCF

(20) Receives "busy" message from SSF

(21) Formats message to MCF (disconnect A)

(22) Receives and analyzes reply from MCF

(23) Formats trunk status message to OMF

(24) Sends message to SSF (send "busy" tone to A)

(25) Receives message from SSF (A goes on-hook)

(26) Sends message to SSF (stop "busy" to A)

(27) Sends message to OMF (circuit usage report)

(28) Update tables

Trunk-to-Local

(1) Receives "call initiate" message from SSF

(2) Analyzes message, performs routing (digit translation)

(3) Analyzes classmark of called terminal

(4) Sends "acknowledge" message to SSF

(5) Formats message to MCF (connect trunk to A)

(6) Receives and analyzes reply from MCF

(7) Formats "busy" terminal message to SSF

(8) Receives "acknowledge" from SSF

(9) Sends message to OMF (circuit usage report)

(10) Update tables

Trunk-to-Trunk

(1) Receives "call initiate" message from SSF

(2) Sends "acknowledge" to SSF

(3) Analyzes message, performs routing

(4) Examines trunk status tables, selects outgoing trunk

(5) Formats "call initiate" message to SSF

(6) Receives "acknowledgement" from SSF

(7) Formats message to MCF (connect trunk A to trunk B)

(8) Receives and analyzes reply from MCF

(9) Send status message to OMF (trunk usage)

(10) Receives "busy" message from SSF

(11) Sends "acknowledge" message to SSF

(12) Sends "busy" message to SSF

(13) Receives "acknowledge" message from SSF

(14) Formats message to MCF (disconnect trunk A)

(15) Receives and analyzes reply from MCF

(16) Sends "release" message to SSF

(17) Receives "acknowledge" message from SSF

(18) Send status message to OMF (trunk usage update)

(19) Update tables

1.2.4.2 CPF Operations (Digital)

<u>Local-to-Local</u>

(1) Receives request for service from SSF

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (connect A to LKG #1)

(4) Receives and analyzes reply from MCF

(5) Sends message to OMF (LKG usage report)

(6) Formats message to MCF (connect LKG #1 to ASF port)

(7) Receives and analyzes reply from MCF

(8) Sends message to OMF (ASF usage report)

(9) Formats message to SEF (request sync procedure)

(10) Receives "acknowledgement" from SEF

(11) Formats message to ASF

(12) Receives and analyzes reply from ASF (receives digits)

(13) Analyzes digits, performs digit translation

(14) Analyzes classmark of terminal B

(15) Formats message to MCF (connect B to LKG #2)

(16) Receives and analyzes reply from MCF

(17) Formats message to OMF (LKG #2 busy)

(18) Formats message to MCF (connect LKG #2 to ASF port)

(19) Receives and analyzes reply

(20) Formats message to ASF

(21) Formats message to OMF (ASF usage report)

(22) Formats message to SEF (sync request)

(23) Sends message to ASF (ring coordination)

(24) Receives message from ASF

(25) Formats message to MCF (disconnect ASF from LKG #1)

(26) Receives and analyzes reply from MCF

(27) Formats message to OMF (ASF port usage update)

(28) Formats message to MCF (disconnect ASF from LKG #2)

(29) Receives and analyzes reply from MCF

(30) Formats message to OMF (ASF port usage update)

(31) Formats message to MCF (connect LKG #1 to LKG #2)

(32) Receives and analyzes reply from MCF

(33) Formats message to SEF (key transfer & sync)

(34) Receives "acknowledgement" from SEF

(35) Formats message to MCF (disconnect LKG #1)

(36) Receives and analyzes reply from MCF

(37) Sends message to OMF (LKG #1 usage update)

(38) Formats message to MCF (disconnect LKG #2)

(39) Receives and analyzes reply from MCF

(40) Formats message to MCF (connect A to B)

(41) Receives and analyzes reply from MCF

(42) Formats message to OMF (LKG #2 usage update)

(43) Receives release information from SSF (A or B releases)

(44) Formats message to MCF (disconnect A or B)

(45) Receives and analyzes reply from MCF

(46) Analyzes classmark of remaining terminal

(47) Formats report message to OMF (usage update)

(48) Directs SSF to send "busy" to remaining party

(49) Receives message from SSF (remaining party release)

(50) Directs SSF to stop "busy" tone

(51) Update status table

<u>Local-to-Trunk</u>

(1) Receives request for service from SSF (A goes off-hook)

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (connect A to ASF port)

(4) Receives and analyzes reply from MCF

(5) Formats message to ASF

(6) Formats message to OMF (ASF usage report)

(7) Receives and analyzes reply from ASF (receives digits)

(8) Formats message to MCF (disconnect A from ASF port)

(9) Sends message to OMF (ASF usage update)

(10) Receives and analyzes reply from MCF

(11) Analyzes digits, performs digit translation

(12) Performs routing, determines primary trunk group

(13) Examines trunk status table, selects idle trunk, marks trunk

(14) Analyzes classmark of trunk

(15) Formats "call initiate" common channel message, sends to SSF

(16) Receives confirmation from SSF

(17) Formats message to OMF (trunk usage message)

(18) Formats message to MCF (connect A to trunk)

(19) Receives and analyzes reply from MCF

(20) Receives "call complete" message from SSF

(21) Receives message from SSF (A goes on-hook)

(22) Formats message to MCF (disconnect A)

(23) Receives and analyzes reply from MCF

(24) Formats common channel (release) message, send to SSF

(25) Receives confirmation of trunk release from SSF

(26) Formats message to OMF (trunk usage update)

(27) Update status tables

Trunk-to-Local

(1) Receives "call initiate" message from SSF

(2) Analyzes message, performs routing (digit translation)

(3) Analyzes classmark of called terminal

(4) Sends "acknowledge" message to SSF

(5) Formats message to MCF (connect trunk to LKG #1)

(6) Receives and analyzes reply from MCF

(7) Formats message to OMF (LKG #1 usage)

(8) Formats message to MCF (connect LKG #1 to ASF port)

(9) Receives and analyzes reply from MCF

(10) Formats message to ASF

(11) Formats message to OMF (ASF port usage)

(12) Formats message to MCF (connect A to LKG #2)

(13) Receives and analyzes reply

(14) Formats message to OMF (LKG #2 usage)

(15) Formats message to MCF (connect LKG #2 to ASF port)

(16) Receives and analyzes reply from MCF

(17) Formats message to OMF (ASF usage report)

(18) Formats message to ASF (ring coordination)

(19) Formats message to SEF (sync request)

(20) Receives message from ASF

(21) Formats message to MCF (disconnect LKG #1 from ASF)

(22) Receives and analyzes reply from MCF

(23) Formats message to OMF (ASF usage update)

(24) Formats message to MCF (disconnect LKG #2 from ASF)

(25) Receives and analyzes reply from MCF

(26) Formats message to OMF (ASF usage update)

(27) Formats message to MCF (connect LKG #1 to LKG #2)

(28) Receives and analyzes reply from MCF

(29) Formats message to SEF (key transfer & sync)

(30) Receives "acknowledge" from SEF

(31) Formats message to MCF (disconnect LKG #1)

(32) Receives and analyzes reply from MCF

(33) Formats message to OMF (LKG #1 usage update)

(34) Formats message to MCF (disconnect LKG #2)

(35) Receives and analyzes reply from MCF

(36) Formats message to OMF (LKG #2 usage update)

(37) Formats message to MCF (connect trunk to A)

(38) Receives and analyzes MCF reply

(39) Formats "call complete" message to SSF

(40) Receives "acknowledge" from SSF

(41) Receives "release" message from SSF

(42) Send "acknowledge" message to SSF

(43) Formats message to MCF (disconnect trunk)

(44) Receives and analyzes reply from MCF

(45) Formats message to SSF (send busy)

(46) Receives release message from SSF

(47) Sends message to SSF (stop "busy")

(48) Formats message to OMF (usage update)

(49) Update tables

<u>Trunk-to-Trunk</u>

(1) Receives "call initiate" message from SSF

(2) Sends "acknowledge" to SSF

(3) Analyzes message, performs routing

(4) Examines trunk status tables, selects outgoing trunk

(5) Analyzes classmark of trunk

(6) Formats "call initiate" message to SSF

(7) Receives "acknowledge" from SSF

(8) Formats message to MCF (connect trunk A to trunk B)

(9) Receives and analyzes reply from MCF

(10) Send status message to OMF (trunk usage)

(11) Receives "call complete" message from SSF

(12) Sends "acknowledge" message to SSF

(13) Sends "call complete" message to SSF

(14) Receives "acknowledge" message from SSF

(15) Receives "release" message from SSF

(16) Send "acknowledge" message from SSF

(17) Formats message to MCF (disconnect Trunk A)

(18) Receives and analyzes reply from MCF

(19) Sends "release" message to SSF

(20) Receives "acknowledge" message from SSF

(21) Send status message to OMF

(22) Update tables

<u>Call Busy Factor Adder (Digital)</u>

<u>Local-to-Local</u>

(1) Receives request for service from SSF

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (connect A to LKG #1)

(4) Receives and analyzes reply from MCF

(5) Sends message to OMF (LKG #1 usage report)

(6) Formats message to MCF (connect LKG #1 to ASF port)

(7) Receives and analyzes reply from MCF

(8) Formats message to SEF (request sync procedure)

(9) Receives "acknowledgement" from SEF

(10) Formats message to ASF

(11) Formats message to OMF (ASF usage report)

(12) Receives and analyzes reply from ASF (receives digits)

(13) Analyzes digits, performs digit translation

(14) Analyzes classmark of terminal B

(15) Formats message to MCF (connect B to LKG #2)

(16) Receives and analyzes reply from MCF (returns "B busy" message)

(17) Formats message to MCF (disconnect LKG #1)

(18) Receives and analyzes reply from MCF

(19) Formats message to OMF (LKG usage update)

(20) Formats message to MCF (disconnect A)

(21) Receives and analyzes reply from MCF

(22) Formats report message to OMF

(23) Directs SSF to send "busy" to A

(24) Receives message from SSF (A releases)

(25) Sends message to SSF (stop "busy")

(26) Sends message to OMF (circuit usage report)

(27) Update tables

Local-to-Trunk

(1) Receives request for service from SSF (A goes off-hook)

(2) Analyzes classmark of requesting terminal (determine signaling convention)

(3) Formats message to MCF (connect A to ASF port)

(4) Receives and analyzes reply from MCF

(5) Formats message to ASF

(6) Formats ASF unit usage report to OMF

(7) Receives and analyzes reply from ASF (receives digits)

(8) Formats message to MCF (disconnect A from ASF port)

(9) Sends usage message to OMF

(10) Receives and analyzes reply from MCF

(11) Analyzes digits, performs digit translation

(12) Performs routing, determines primary trunk group

(13) Examines trunk status table, selects idle trunk, marks trunk

(14) Analyzes classmark of trunk

(15) Formats "call initiate" common channel message, sends to SSF

(16) Receives confirmation from SSF

(17) Formats message to OMF (trunk usage message)

(18) Formats message to MCF (connect A to trunk)

(19) Receives and analyzes reply from MCF

(20) Receives "busy" message from SSF

(21) Formats message to MCF (disconnect A)

(22) Receives and analyzes reply from MCF

(23) Formats common channel acknowledge message, send to SSF

(24) Receives confirmation of trunk release from SSF

(25) Formats trunk status message to OMF

(26) Update status tables

Trunk-to-Local

(1) Receives "call initiate" message from SSF

(2) Analyzes message, performs routing (digit translation)

(3) Analyzes classmark of called terminal

(4) Sends "acknowledge" message to SSF

(5) Formats message to MCF (connect trunk to LKG #1)

(6) Receives and analyzes reply from MCF

(7) Formats message to OMF (LKG #1 usage)

(8) Formats message to MCF (connect LKG #1 to ASF port)

(9) Receives and analyzes reply from MCF

(10) Formats message to ASF

(11) Formats message to OMF (ASF port usage)

(12) Formats message to MCF (connect A to LKG #2)

(13) Receives and analyzes reply (returns "B busy" message)

(14) Formats "busy" message to SSF

(15) Receives "acknowledge" from SSF

(16) Formats message to MCF (disconnect trunk)

(17) Receives and analyzes reply

(18) Sends message to OMF (trunk usage update)

(19) Formats message to MCF (disconnect LKG #1)

(20) Receives and analyzes reply

(21) Sends message to OMF (LKG #1 usage update)

(22) Sends message to OMF (ASF usage update)

(23) Updates tables

Trunk-to-Trunk

(1) Receives "call initiate" message from SSF

(2) Sends "acknowledge" to SSF

(3) Analyzes message, performs routing

(4) Examines trunk status tables, selects outgoing trunk

(5) Analyzes classmark of trunk

(6) Formats "call initiate" message to SSF

(7) Receives "acknowledgement" from SSF

(8) Send status message to OMF (trunk usage)

(9) Receives "busy" message from SSF

(10) Sends "acknowledge" message to SSF

(11) Sends "busy" message to SSF

(12) Receives "acknowledge" message from SSF

(13) Receives "release" message from SSF

(14) Send "acknowledge" message from SSF

(15) Sends "release" message to SSF

(16) Receives "acknowledge" message from SSF

(17) Send status message to OMF

(18) Update tables

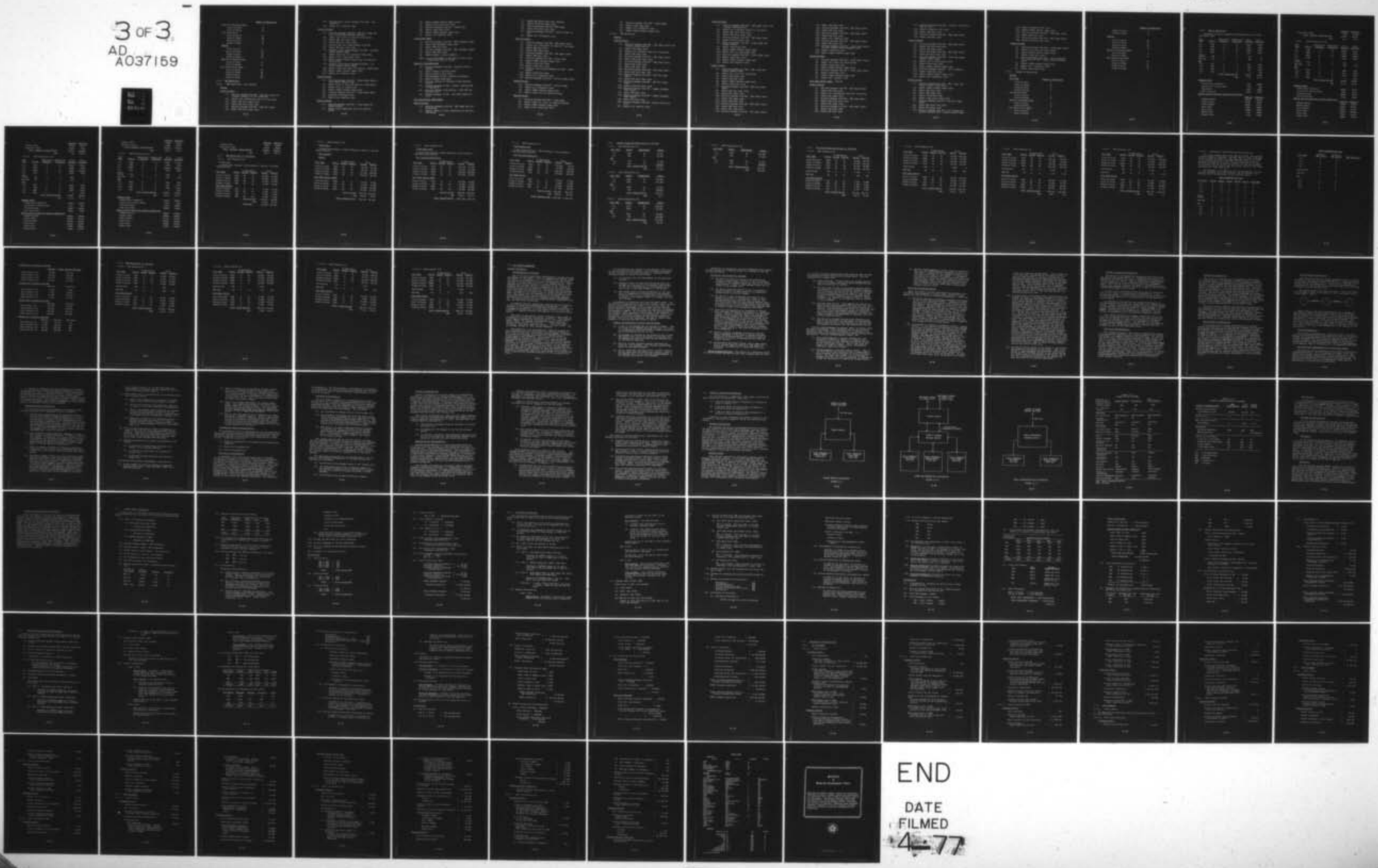
1.2.4.3 Summary of CPF Operations

Analog

Call Type	Number of Operations
Local-to-Local	33
Local-to-Trunk	28
Trunk-to-Local	21
Trunk-to-Trunk	22
Loop-Around Adder	
Local-to-Local	11
Local-to-Trunk	11

AD-A037 159 NORTH ELECTRIC CO GALION OHIO F/G 17/2
COMMUNICATIONS PROCESSOR SYSTEM.(U)
JAN 77 K HAGSTROM, B BEIZER F30602-73-C-0314
UNCLASSIFIED RADC-TR-76-394 NL

3 OF 3
AD
A037159



END
DATE
FILMED
4-77

	Number of Operations
Operator Forwarded Adder	
Local-to-Local	13
Local-to-Trunk	13
Call Busy Factor	
Local-to-Local	19
Local-to-Trunk	28
Trunk-to-Local	10
Trunk-to-Trunk	19
Digital	
Local-to-Local	51
Local-to-Trunk	27
Trunk-to-Local	49
Trunk-to-Trunk	22
Operator Forwarded Adder	
Local-to-Local	13
Local-to-Trunk	13
Call Busy Factor Adder	
Local-to-Local	27
Local-to-Trunk	26
Trunk-to-Local	23
Trunk-to-Trunk	18

1.2.5 OMF Operations

1.2.5.1 OMF Operations - Call Related

Analog

Local-to-Local

(1) Receives message from CPF - ASF unit usage and terminal identity of call originator

(2) Update call originating count for line group

(3) Update ASF unit usage count

(4) Update ASF units busy count

(5) Receives message from CPF - ASF unit usage

(6) Update ASF units busy count

(7) Receives busy circuit message from CPF - call connected

(8) Update call complete count

Local-to-Trunk

(1) Receives message from CPF - ASF unit usage and call originating terminal identity

(2) Update call initiate count for line group

(3) Update ASF unit busy count

(4) Update ASF unit usage count

(5) Receives ASF unit usage message from CPF

(6) Update ASF unit busy count

(7) Receives trunk usage message from CPF - includes common channel usage

(8) Update busy count for specific trunk group

(9) Update total trunk usage count

(10) Update common channel usage count for specific trunk group

(11) Receives trunk status message from CPF - includes common channel usage

(12) Update trunk busy count for specific trunk group

(13) Update common channel usage count

(14) Update call complete count

Trunk-to-Local

(1) Receives message from CPF - trunk usage update - also common channel usage

(2) Update count of calls for specific trunk group

(3) Update count of trunks busy

(4) Update common channel usage count

(5) Receives message from CPF - trunk usage update

(6) Decrease trunk busy count

(7) Update call complete count

Trunk-to-Trunk

(1) Receives message from CPF - trunk usage and common channel

(2) Update trunk usage busy count for specific groups

(3) Update common channel usage counts

(4) Update total trunk call count

(5) Receives message from CPF - trunks idle

(6) Update trunk busy count

(7) Update common channel usage count

(8) Update call complete count

<u>Loop-Around Adder</u>

(1) Receives message from CPF - SEF equipment usage

(2) Update SEF equipment usage count

(3) Update SEF busy count

(4) Receives message from CPF - SEF equipment update

(5) Update SEF busy count

(6) Update number of calls complete

NOTE: Loop-around adder is the same for both Local-to-Local and Local-to-Trunk.

<u>Operator Forwarded Adder</u>

(1) Receives message from CPF - operator service request

(2) Places request in proper queue

(3) Update operator call count

(4) Send messages to CPF - operator assignment

(5) Remove request from queue

(6) Receives address of terminal B from operator console

(7) Formats messages to CPF - connect outgoing OPR to terminal B

(8) Receives message from operator - drop OPR connection A & B

(9) Formats messages to CPF - drop OPR connection A & B

<u>Call Busy Factor (CBF) Adder</u>

<u>Local-to-Local</u>

(1) Receives messages from CPF - ASF usage and call originating

(2) Updates number of calls originated for specific line group

(3) Update ASF ports busy count terminal

(4) Update ASF ports usage count

(5) Receives message from CPF (ASF usage)

(6) Update ASF ports busy count

(7) Receives message from CPF - circuit usage report

(8) Update call incomplete count

<u>Local-to-Trunk</u>

(1) Receives message from CPF - ASF usage report

(2) Update call originated for specific line group

(3) Update ASF busy count

(4) Update ASF usage count

(5) Receives message from CPF - ASF usage report

(6) Update ASF busy count

(7) Receives message from CPF - trunk usage

(8) Update common channel usage count

(9) Update trunk busy count

(10) Update trunk call count

(11) Receives trunk status message from CPF - trunk status

(12) Update trunk busy count

(13) Update call incomplete count

(14) Receives message from CPF - circuit usage report

<u>Trunk-to-Local</u>

(1) Receives message from CPF - circuit usage

(2) Update call incomplete count

(3) Update common channel usage count

(4) Update trunk call for specific group

<u>Trunk-to-Trunk</u>

(1) Receives message from CPF - trunk usage

(2) Update trunk call count for specific groups

(3) Update common channel usage count

(4) Update total trunk busy count

(5) Receives message from CPF - trunk usage

(6) Update trunk busy count

(7) Update call incomplete count

(8) Update common channel usage count

1.2.5.2 OMF Operations

<u>Digital</u>

<u>Local-to-Local</u>

(1) Receives message from CPF - LKG usage report and terminal identification

(2) Update LKG usage count

(3) Update call originate count for line group

(4) Update LKG busy count

(5) Receives message from CPF - ASF usage report

(6) Update ASF ports busy count

(7) Receives message from CPF - LKG usage report

(8) Update LKG usage count

(9) Update LKG busy count

(10) Receives message from CPF - ASF usage report

(11) Update ASF unit busy count

(12) Receives message from CPF - ASF port usage report

(13) Update ASF ports busy count

(14) Receives message from CPF - ASF port usage report

(15) Update ASF ports busy count

(16) Receives message from CPF - common equipment (LKG) report

(17) Update LKG busy count

(18) Receives message from CPF - common equipment (LKG) report

(19) Update LKG busy count

(20) Receives message from CPF - matrix connection count

(21) Update call complete count

Local-to-Trunk

(1) Receives message from CPF - ASF usage report and terminal identification

(2) Update call originate count for line group

(3) Update ASF ports busy count

(4) Update ASF ports usage count

(5) Receives message from CPF - ASF usage report

(6) Update ASF port busy count

(7) Receives message from CPF - trunk usage and common channel

(8) Update trunk busy count

(9) Update common channel usage count

(10) Update call to trunk group count

(11) Receives message from CPF - trunk usage report

(12) Update trunk busy count

(13) Update common channel usage count

(14) Update call complete count

Trunk-to-Local

(1) Receives message from CPF - LKG, trunk and common channel usage

(2) Update call count for trunk group

(3) Update LKG usage count

(4) Update LKG busy count

(5) Update common channel usage count

(6) Update trunk busy count

(7) Receives message from CPF - ASF port usage report

(8) Update ASF port busy count

(9) Update ASF usage count

(10) Receives message from CPF - LKG usage report

(11) Update LKG usage count

(12) Update LKG busy count

(13) Receives message from CPF - ASF usage report

(14) Update ASF busy count

(15) Update ASF usage count

(16) Receives message from CPF - ASF usage report

(17) Update ASF busy count

(18) Receives message from CPF - ASF usage report

(19) Update ASF busy count

(20) Receives message from CPF - LKG usage report

(21) Update LKG busy count

(22) Update call complete count

(23) Receives message from CPF - LKG usage report

(24) Update LKG busy count

(25) Receives message from CPF - trunk usage report - includes common channel message count

(26) Update trunk busy count

(27) Update common channel usage count

<u>Trunk-to-Trunk</u>

(1) Receives message from CPF - trunk usage report

(2) Update call count for trunk groups

(3) Update trunk busy count

(4) Update common channel usage count

(5) Receives message from CPF - trunk usage report

(6) Update trunk busy count

(7) Update common channel usage count

(8) Update call complete count

<u>Call Busy Factor Adder</u> - Digital

<u>Local-to-Local</u>

(1) Receives message from CPF - LKG usage report

(2) Update LKG busy count

(3) Update call origination count for line group

(4) Update LKG usage count

(5) Receives message from CPF - ASF usage report

(6) Update ASF ports busy count

(7) Update ASF usage count

(8) Receives message from CPF - LKG usage report

(9) Update LKG usage count

(10) Update LKG busy count

(11) Receives message from CPF - "B busy", therefore, LKG now idle

(12) Update incomplete call count

(13) Update LKG busy count

(14) Receives message from CPF - LKG usage report

(15) Update LKG busy count

Local-to-Trunk

(1) Receives message from CPF - ASF usage report

(2) Update ASF port busy count

(3) Update ASF usage count

(4) Receives message from CPF - ASF usage report

(5) Update ASF port busy count

(6) Receives message from CPF - trunk and common channel usage report

(7) Update call to trunk group count

(8) Update trunk busy count

(9) Update common channel usage count

(10) Receives message from CPF - call incomplete

(11) Update trunk busy count

(12) Update common channel usage count

(13) Update call incomplete count

Trunk-to-Local

(1) Receives message from CPF - LKG, trunk, and common channel usage report

(2) Update common channel usage count

(3) Update LKG busy count

(4) Update LKG usage count

(5) Update trunk busy count

(6) Update trunk group usage count

(7) Receives message from CPF - ASF port usage report

(8) Update ASF ports busy count

(9) Update ASF usage count

(10) Receives message from CPF - call incomplete because terminal busy - common channel usage

(11) Update incomplete call count
(12) Update common channel usage count
(13) Receives message from CPF - LKG usage report
(14) Update LKG busy count
(15) Receives message from CPF - ASF usage report
(16) Update ASF ports busy count

<u>Trunk-to-Trunk</u>

(1) Receives message from CPF - trunk usage report
(2) Update call count for trunk groups
(3) Update common channel usage count
(4) Update trunk busy count
(5) Receives message from CPF - call incomplete, two trunks now idle, common channel message count
(6) Update common channel usage count
(7) Update call incomplete count
(8) Update trunk busy count

1.2.5.3 Summary of Operations

<u>Analog</u>

Call Type	Number of Operations
Local-to-Local	8
Local-to-Trunk	14
Trunk-to-Local	7
Trunk-to-Trunk	8
Loop-Around Adder	
Local-to-Local	6
Local-to-Trunk	6
Operator Forwarded Adder	
Local-to-Local	9
Local-to-Trunk	9
Call Busy Factor Adder	
Local-to-Local	8
Local-to-Trunk	14

	Number of Operations
Trunk-to-Local	4
Trunk-to-Trunk	8
Digital	
Local-to-Local	21
Local-to-Trunk	14
Trunk-to-Local	27
Trunk-to-Trunk	8
Operator Forwarded Adder	
Local-to-Local	9
Local-to-Trunk	9
Call Busy Factor Adder	
Local-to-Local	15
Local-to-Trunk	13
Trunk-to-Local	16
Trunk-to-Trunk	8

1.2.6 Matrix Operations

Breakdown of Matrix Commands and Operations by call type and system size.

Call Type	#Calls	#Operations (Analog)	#Operations (Digital)	Total (Analog)	Total (Digital)
L-L	1080	8	16	8640	17280
L-T	1080	4	4	4320	4320
T-L	1320	6	16	7920	21120
T-T	1320	2	2	2640	2640
Loop-Around	108	4	0	432	0
OPR Fwd.	24	4	4	96	96
CBF					
L-L	270	3	5	810	1350
L-T	270	2	4	540	1080
T-L	330	1	5	330	1650
T-T	330	0	2	0	660
		TOTAL OPERATIONS/BH		25728	50196
			/SEC	7.15	13.94

Command Types		
Connections (complete)	7764	19998
Connections (return busy)	600	600
Disconnect	7764	19998
Tone Injection	9600	9600

Matrix Control Function Internal Operations

	(Analog)	(Digital)
Command Decode	25728	50196
Format Reply	25728	50196
Find Terminal	32256	81192
Trace Path	32892	69594
Store Path	15528	39996
Search Path	7764	19998
Connect Path	7764	19998

	(Analog)	(Digital)
Disconnect Path	7764	19998
TOTAL INTERNAL OPERATIONS/BH	155424	351168
/SEC	43.2	97.5

1.2.6.1 2400 Termination C/S

Call Type	#Calls	#Operations (Analog)	#Operations (Digital)	Total (Analog)	Total (Digital)
L-L	4032	8	16	32256	64512
L-T	2688	4	4	10752	10752
T-L	6048	6	16	36288	96768
T-T	4032	2	2	8064	8064
Loop-Around	336	4	0	1344	0
OPR Fwd.	68	4	4	272	272
CBF					
L-L	1008	3	5	3024	5040
L-T	672	2	4	1344	2688
T-L	1512	1	5	1512	7560
T-T	1008	0	2	0	2016
		TOTAL OPERATIONS/BH		94856	197672
			/SEC	26.35	60.35

Command Types		
Connections (Complete)	26008	77416
Connections (Return Busy)	2520	2520
Disconnections	26008	77416
Tone Injections	40320	40320

Matrix Control Function Internal Operations	(Analog)	(Digital)
Command Decode	94856	197672
Format Reply	94856	197672
Find Terminal	109072	314704
Trace Path	118344	272568
Store Path	52016	154832
Search Path	26008	77416

	(Analog)	(Digital)
Connect Path	26008	77416
Disconnect Path	26008	77416
TOTAL INTERNAL OPERATIONS/BH	547168	1369696
/SEC	152	380.47

1.2.6.2 4200 Termination C/S

Call Type	#Calls	#Operations (Analog)	#Operations (Digital)	Total (Analog)	Total (Digital)
L-L	6174	8	16	49392	98784
L-T	2645	4	4	10580	10580
T-L	11466	6	16	68796	183456
T-T	4914	2	2	9828	9828
Loop-Around	441	4	0	1764	0
OPR Fwd.	89	4	4	356	
CBF					
L-L	1544	3	5	4632	7720
L-T	662	2	4	1324	2648
T-L	2867	1	5	2867	14335
T-T	1229	0	2	0	2458
		TOTAL OPERATIONS/BH		149539	330165
			/SEC	41.54	91.71

Command Types		
Connections (complete)	37284	127597
Connections (return busy)	4411	4411
Disconnections	37284	127597
Tone Injections	70560	70560
Matrix Control Function Internal Operations		
Command Decode	149539	330165
Format Reply	149539	330165
Find Terminal	157958	519210
Trace Path	182412	453351
Store Path	74568	255194
Search Path	37284	127597

	(Analog)	(Digital)
Connect Path	37284	127597
Disconnect Path	37284	127597
TOTAL INTERNAL OPERATIONS/BH	825868	1707459
/SEC	229.4	474.3

1.2.6.3 6000 Termination C/S

Call Type	#Calls	#Operations (Analog)	#Operations (Digital)	Total (Analog)	Total (Digital)
L-L	7200	8	16	57600	115200
L-T	1800	4	4	7200	7200
T-L	16800	6	16	100800	268800
T-T	4200	2	2	8400	8400
Loop-Around	450	4	0	1800	0
OPR Fwd.	90	4	4	360	360
CBF					
L-L	1800	3	5	5400	9000
L-T	450	2	4	900	1800
T-L	4200	1	5	4200	21000
T-T	1050	0	2	0	2100
		TOTAL OPERATIONS/BH		186660	433860
			/SEC	51.85	120.52

Command Types		
Connections (complete)	42330	165930
Connections (return Busy)	6000	6000
Disconnections	42330	165930
Tone Injections	96000	96000
Matrix Control Function Internal Operations		
Command Decode	186660	433860
Format Reply	186660	433860
Find Terminal	181320	675720
Trace Path	222990	593790
Store Path	84660	331860
Search Path	42330	165930

	(Analog)	(Digital)
Connect Path	42330	165930
Disconnect Path	42330	165930
TOTAL INTERNAL OPERATIONS/BH	989280	2966880
/SEC	274.8	824.13

1.2.7 SSF Operations vs. C/S Sizes

1.2.7.1 600 Terminal C/S

Fixed type

3 operational terminal x 600 terminal x 100/sec = 180,000 operations/sec

		# Operations		Total	
Call Type	#Calls	Analog	Digital	Analog	Digital
Local-to-Local	1080	24	10	25,920	10,800
Local-to-Trunk	1080	21	16	22,680	17,280
Trunk-to-Local	1320	25	18	33,000	23,760
Trunk-to-Trunk	1320	24	24	31,680	31,680
Call Busy Factor					
Local-to-Local	270	15	8	4,050	2,160
Local-to-Trunk	270	22	10	5,940	2,700
Trunk-to-Local	330	8	8	2,640	2,640
Trunk-to-Trunk	330	20	24	6,600	7,940
		Operation/BH		132,510	98,960
			/SEC	36.81	27.49
		TOTAL/SEC		180,037	180,027

1.2.7.2   2400 Terminal C/S

Fixed Type

3 operations/terminal x 2400 terminals x 100/sec = 720,000 operations/sec

Call Related

Analog

Call Type	#Calls	# Operations Analog	# Operations Digital	Total Analog	Total Digital
Local-to-Local	4032	24	10	96,768	40,320
Local-to-Trunk	2688	21	16	56,448	43,008
Trunk-to-Local	6048	25	18	151,200	108,864
Trunk-to-Trunk	4032	25	24	100,800	96,768
Call Busy Factor Adder					
Local-to-Local	1008	15	8	15,120	8,064
Local-to-Trunk	672	22	10	14,784	6,720
Trunk-to-Local	1512	8	8	12,096	12,096
Trunk-to-Trunk	1008	20	24	20,160	24,192
			Operations/BH	467,376	340,032
			/SEC	129.83	94.45
		TOTAL OPERATION/SEC		720,130	720,094

1.2.7.3 4200 Terminal C/S

Fixed Operations

3 operations/terminal x 4200 terminals x 100 scan/sec = 1,260,000 operations/sec

Call Releated Operations

		# Operations		Total	
Call Type	#Calls	Analog	Digital	Analog	Digital
Local-to-Local	6174	24	10	148,176	61,740
Local-to-Trunk	2645	21	16	55,545	42,320
Trunk-to-Local	11466	25	18	286,650	206,388
Trunk-to-Trunk	4914	24	24	117,936	117,936
Call Busy Factor Adder					
Local-to-Local	1544	15	8	23,160	12,352
Local-to-Trunk	622	22	10	13,684	6,220
Trunk-to-Local	2867	8	8	22,936	22,936
Trunk-to-Trunk	1229	20	24	24,580	29,496
		Operations/BH		692,667	499,388
			/SEC	192.41	138.72
		TOTAL OPERATION/SEC		1,260,192	1,260,139

1.2.7.4 6000 Terminal C/S

Fixed Operations

3 operations/terminal x 6000 terminals x 100 scan/sec = 1,800,000 operations/sec

Call Related Operations

		# Operations		Total	
Call Type	#Calls	Analog	Digital	Analog	Digital
Local-to-Local	7200	24	10	172,800	72,000
Local-to-Trunk	1800	21	16	37,800	28,800
Trunk-to-Local	16800	25	18	420,000	302,400
Trunk-to-Trunk	4200	24	24	100,800	100,800
Call Busy Factor					
Local-to-Local	1800	15	8	27,000	14,400
Local-to-Trunk	450	22	10	9,900	4,500
Trunk-to-Local	4200	8	8	33,600	33,600
Trunk-to-Trunk	1050	20	24	21,000	25,200
		Operations/BH		822,900	581,700
			/SEC	228.58	161.58
		TOTAL OPERATION/SEC		1,800,229	1,800,162

1.2.8 Address Signaling Operations vs. C/S Size

1.2.8.1 600 Termination C/S

CALL TYPE	#CALLS	#OPERATIONS	TOTAL
L-L	1080	34	36,720
L-T	1080	51	55,080
CBF			
L-L	270	34	9,180
L-T	270	51	13,770
	TOTAL OPERATIONS/BH		114,750
		/SEC	31.875

1.2.8.2 2400 Termination C/S

CALL TYPE	#CALLS	#OPERATIONS	TOTAL
L-L	4032	34	137,088
L-T	2688	51	137,088
CBF			
L-L	1008	34	34,272
L-T	672	51	34,272
	TOTAL OPERATIONS/BH		342,720
		/SEC	95.2

1.2.8.3 4200 Termination C/S

CALL TYPE	#CALLS	#OPERATIONS	TOTAL
L-L	6174	34	209,916
L-T	2645	51	134,895
CBF			
L-L	1544	34	52,496
L-T	662	51	33,762
	TOTAL OPERATIONS/BH		431,069
		/SEC	119.74

1.2.8.4 6000 Termination C/S

CALL TYPE	#CALLS	#OPERATIONS	TOTAL
L-L	7200	34	244,800
L-T	1800	51	91,800
CBF			
L-L	1800	34	61,200
L-T	450	51	22,950
	TOTAL OPERATIONS/BH		420,750
		/SEC	116.875

1.2.9 Call Processing Operations vs. C/S Size

1.2.9.1 600 Terminal C/S

		# Operations		Total	
Call Type	#Calls	Analog	Digital	Analog	Digital
Local-to-Local	1080	33	51	35,640	55,080
Local-to-Trunk	1080	28	27	30,240	29,160
Trunk-to-Local	1320	21	49	27,720	64,680
Trunk-to-Trunk	1320	22	22	29,040	29,040
Loop-Around	108	11	0	1,188	0
OPR Fwd.	24	13	13	312	312
Call Busy Factor					
Local-to-Local	270	19	27	5,130	7,290
Local-to-Trunk	270	28	26	7,560	7,020
Trunk-to-Local	330	10	23	3,300	7,590
Trunk-to-Trunk	330	19	18	6,270	5,940
	TOTAL OPERATIONS/BH			146,400	206,112
			/SEC	40.67	57.25

1.2.9.2 2400 Terminal C/S

		# Operations		Total	
Call Type	#Calls	Analog	Digital	Analog	Digital
Local-to-Local	4032	33	51	133,056	205,632
Local-to-Trunk	2688	28	27	75,264	75,264
Trunk-to-Local	6048	21	49	127,008	127,008
Trunk-to-Trunk	4032	22	22	88,704	88,704
Loop-Around	336	11	0	3,696	0
OPR Fwd.	68	13	13	884	884
Call Busy Factor					
Local-to-Local	1008	19	27	19,152	27,216
Local-to-Trunk	672	28	26	18,816	17,472
Trunk-to-Local	1512	10	23	15,120	34,776
Trunk-to-Trunk	1008	19	18	19,152	18,144
	TOTAL OPERATIONS/BH			500,852	595,100
			/SEC	139.13	165.31

1.2.9.3 4200 Terminal C/S

Call Type	#Calls	# Operations Analog	# Operations Digital	Total Analog	Total Digital
Local-to-Local	6174	33	51	203,742	314,874
Local-to-Trunk	2645	28	27	74,060	74,060
Trunk-to-Local	11466	21	49	240,786	561,834
Trunk-to-Trunk	4914	22	22	108,108	108,108
Loop-Around	441	11	0	4,851	0
OPR Fwd.	89	13	13	1,157	1,157
Call Busy Factor					
Local-to-Local	1544	19	27	29,336	41,688
Local-to-Trunk	662	28	26	18,536	17,212
Trunk-to-Local	2867	10	23	28,670	65,941
Trunk-to-Trunk	1229	19	18	23,351	22,122
	TOTAL OPERATIONS/BH			732,597	1,205,839
			/SEC	203.5	335

1.2.9.4 6000 Terminal C/S

Call Type	#Calls	# Operations Analog	# Operations Digital	Total Analog	Total Digital
Local-to-Local	7200	33	51	237,600	367,200
Local-to-Trunk	1800	28	27	50,400	48,600
Trunk-to-Local	16800	21	49	352,800	823,200
Trunk-to-Trunk	4200	22	22	92,400	92,400
Loop-Around	450	11	0	4,950	0
OPR Fwd.	90	13	13	1,170	1,170
Call Busy Factor					
Local-to-Local	1800	19	27	34,200	48,600
Local-to-Trunk	450	28	26	12,600	11,700
Trunk-to-Local	4200	10	23	42,000	96,600
Trunk-to-Trunk	1050	19	18	18,900	18,900
	TOTAL OPERATIONS/BH			847,020	1,508,370
			/SEC	235.3	419

1.2.9.5 Breakdown of Inter-function Data Transfers

For every message sent by the CPF to the MCF, ASF and SSF, a reply message is required. The CPF, when it sends a message, starts a timer and stops the timer when the reply is received. It is implied that part of the CPF overhead function is the periodic checking of the timer to detect time-out conditions. In this way, the CPF is able to exercise its overall coordination task and keep track of the operational states of the other functions on a real time basis.

The messages to the OMF fall into two categories; (1) statistical messages for record keeping, and (2) operator call messages which the OMF places in the operator queue.

DATA TRANSFERS PER CALL

Call Type	MCF(A)	MCF(D)	ASF(A)	ASF(D)	SSF(C)	SSF(CCSM)
L-L	8	16	1	3	6	0
L-T	4	4	1	1	6	3
T-L	6	16	0	2	3	5
T-T	2	2	0	0	0	12
Loop-Around	4	0	0	0	0	0
OPR Fwd.	4	4	0	0	0	0
CBF						
L-L	3	5	1	1	4	0
L-T	2	4	1	1	4	0
T-L	1	5	0	1	0	4
T-T	0	2	0	0	0	10

DATA TRANSFERS PER CALL

Call Type	(Analog) OMF (Start)	(Digital) OMF (Start)	OMF (OPR Cont)
L-L	4	9	
L-T	4	4	
T-L	2	9	
T-T	2	2	
Loop-Around	2	0	
OPR Fwd.	0	0	4
CBF			
L-L	3	5	
L-T	5	4	
T-L	1	5	
T-T	2	2	

CPF/SSF Data Transfers (per BH)

	Commands	Common Channel Messages
600 Terminal C/S	19,080	30,300
2400 Terminal C/S	65,184	102,816
4200 Terminal C/S	96,136	147,991
6000 Terminal C/S	113,400	161,700

CPF/ASF Data Transfer (per BH)

	(Analog)	(Digital)
600 Terminal C/S	2,700	7,830
2400 Terminal C/S	8,400	30,072
4200 Terminal C/S	11,025	49,172
6000 Terminal C/S	11,250	63,450

CPF/MCF Data Transfer (per BH)

	(Analog)	(Digital)
600 Terminal C/S	25,728	50,196
2400 Terminal C/S	94,856	197,672
4200 Terminal C/S	149,539	330,165
6000 Terminal C/S	186,660	433,860

CPF/OMF Data Transfer (per BH)

	(Analog)	(Digital)	(OPR Cont)
600 Terminal C/S	17,286	33,300	96
2400 Terminal C/S	57,624	126,840	272
4200 Terminal C/S	82,185	206,329	356
6000 Terminal C/S	92,850	265,500	360

1.2.10 OMF Operations vs. C/S Size

1.2.10.1 600 Terminal C/S

Call Type	#Calls	# Operations Analog	# Operations Digital	Total Analog	Total Digital
Local-to-Local	1080	8	21	8,640	22,600
Local-to-Trunk	1080	14	14	15,120	15,120
Trunk-to-Local	1320	7	27	9,240	35,640
Trunk-to-Trunk	1320	8	8	10,560	10,560
Loop-Around	108	6		648	
OPR Fwd.	24	9	9	216	216
Call Busy Factor					
Local-to-Local	270	8	15	2,160	4,050
Local-to-Trunk	270	14	13	3,780	3,510
Trunk-to-Local	330	4	16	1,320	5,280
Trunk-to-Trunk	330	8	8	2,640	2,640
	TOTAL OPERATIONS/BH			54,324	99,616
			/SEC	15.09	27.67

1.2.10.2 2400 Terminal C/S

Call Type	#Calls	# Operations Analog	# Operations Digital	Total Analog	Total Digital
Local-to-Local	4032	8	21	32,256	84,672
Local-to-Trunk	2688	14	14	37,632	37,632
Trunk-to-Local	6048	7	27	42,336	163,296
Trunk-to-Trunk	4032	8	8	32,256	32,256
Loop-Around	336	6		2,016	
OPR Fwd.	68	9	9	612	612
Call Busy Factor					
Local-to-Local	1008	8	15	8,064	15,120
Local-to-Trunk	672	14	13	9,408	8,736
Trunk-to-Local	1512	4	16	6,048	24,192
Trunk-to-Trunk	1008	8	8	8,064	8,064
	TOTAL OPERATIONS/BH			178,692	374,580
			/SEC	49.64	104.05

1.2.10.3 4200 Terminal C/S

Call Type	#Calls	# Operations Analog	# Operations Digital	Total Analog	Total Digital
Local-to-Local	6174	8	21	49,392	129,654
Local-to-Trunk	2645	14	14	37,030	37,030
Trunk-to-Local	11466	7	27	80,262	309,582
Trunk-to-Trunk	4914	8	8	39,312	39,312
Loop-Around	441	6		2,646	
OPR Fwd.	89	9	9	801	801
Call Busy Factor					
Local-to-Local	1544	8	15	12,352	23,160
Local-to-Trunk	662	14	13	9,268	8,606
Trunk-to-Local	2867	4	16	11,468	45,872
Trunk-to-Trunk	1229	8	8	9,832	9,832
	TOTAL OPERATIONS/BH			252,363	646,169
			/SEC	70.10	179.49

1.2.10.4 6000 Terminal C/S

		# Operations		Total	
Call Type	#Calls	Analog	Digital	Analog	Digital
Local-to-Local	7200	8	21	57,600	151,200
Local-to-Trunk	1800	14	14	25,200	25,200
Trunk-to-Local	16800	7	27	117,600	453,600
Trunk-to-Trunk	4200	8	8	33,600	33,600
Loop-Around	450	6		2,700	
OPR Fwd.	90	9	9	810	810
Call Busy Factor					
Local-to-Local	1800	8	15	14,400	27,000
Local-to-Trunk	450	14	13	6,300	5,850
Trunk-to-Local	4200	4	16	16,800	67,200
Trunk-to-Trunk	1050	8	8	8,400	8,400
	TOTAL OPERATIONS/BH			283,410	772,860
			/SEC	78.73	214.68

2.0 M/S SYSTEM PARAMETERS

General Discussion

Accountability Processing

One of the most significant determinants of system cost and performance is the degree to which a communication system, taken as a whole, is accountable, i.e., responsible, for the traffic that has been entrusted to its care. Accountability for traffic delivery is usually conceived as a total network policy. The implementation of that policy is reflected in the totality of hardware and software resources integrated at each individual switching center in the system as well as the topology of low-level and high-level connectivity. Accountability requirements impact strongly on the transit time of traffic through the network, the holding time for messages or associated data at each node, requirements for on-line storage at each node, and requirements for various types of transmission acknowledgements. This is directly reflected in processing delays, additional storage, and additional processing.

In general the most significant computer resource affected is that of storage and acknowledgement processing. A rough estimate of the increase of these from minimum accountability to a maximum accountability is in the order of from 4 to 16 fold increase in computer resource requirements on a switch by switch basis.

The study of the several systems indicated a wide range of accountability intentions (doctrines, or goals). These include: (1) Minimum accountability characterized by "It's your nickel buddy" (ARPA); (2) maximum accountability characterized by "Neither sleet nor hail nor rain nor . . . shall stay these "courriers" from their appointed rounds". In other words - get the message through or "die" trying (AUTODIN).

Accountability is at best seen as a trade-off between the user responsibility (user accountability) and system responsibility for the delivery, and receipt acknowledgement of messages. In a system with minimum accountability a message sender - either human or SDA equipment - would expect to rely on the acknowledgement receipt of the message by a similar message generated by the destined addressee. If no such receipt acknowledgement is received at the message source the options open to the source are few in terms of service support by the communications system. The message may have been lost, the message may have been mis-routed, the message may have been garbled and there is no way through a system service to ascertain exactly what happened to the message. Typically, the message is re-transmitted with a significantly lessened confidence in the successful delivery.

At the opposite end, several of the systems - that is all of the military oriented systems - have incorporated accountability procedures to extreme proportions. Basically these extremes attempt to assure the following:

(1) All messages will be acknowledged for delivery and receipt;

(2) Complete traces of steps and processing given the message at every point of the system is maintained so that any "misadventures" of the message can be made known to the human managers of the system;

(3) The system itself includes procedures for the automatic retransmission or redelivery of the message, generated from its own storage media without having to resort to the message source to re-input the original message.

Accountability intentions fall into two major areas: the system can be accountable for the traffic itself, and/or accountable for the history of the traffic. In the first case the system maintains recordings of the message text at selected points (nodes) in the system and thus can reproduce (retransmit) the message on demand. In the second case the system maintains the necessary event journals to keep track of what happened to the message. While both types of accountability, traffic and historical, are usually provided in most highly accountable systems, they are in fact separable, and may be treated independently at selected switches within a system.

<u>Traffic Accountability Involves the Following</u>

(1) A copy of the message may be obtained on demand. That is a copy of the message awaiting delivery, or a copy of the message already delivered is being retained for some predetermined period of time;

(2) The message is retained by the system (at one or more switching sites) until the system is satisfied that the message has been delivered and has accounted for all deliveries scheduled;

(3) Does not include assurance against duplicate deliveries, lost messages, lost receipts (acknowledgements) or misrouted messages;

(4) For an individual switching center, traffic accountability means that the switch will protect a copy of the message until it has received acknowledgement of all required deliveries of the message.

Historical accountability involves bookkeeping with respect to what happened to a message during its lifetime in the communications system.

Historical Accountability Includes

(1) The switch records on logically and electrically separate storage media, history of the events which occurred to any message during its time in the switch. Traditionally a dozen or more message events are recorded in the journal file;

(2) The switch operating system is able to reproduce a formatted report of these events on a message-by-message basis on operator demand;

(3) The switch is able to display the status of the message in terms of deliverability, when, to whom, the precedence level, waiting time in queue, etc. Historical accountability thus assures against duplicate deliveries, lost deliveries and/or lost receipts by keeping an audit trail of where every message came from and to where every message was delivered;

(4) Historical accountability is considered to demand more stringent processing requirements on a switching complex. While it requires less total storage than storage of the message text itself, significantly more processing activity is required in terms of demand on I/O channels and access to on-line storage media for the recording of the journal records. Historical accountability frequently is considered first in switching system design, over traffic accountability. The journal record files may be retained for a longer period of time than the message history files themselves;

(5) There is usually a minimum of historical accountability designed for HH transit centers utilizing packet switching procedures. In contrast there is frequently no traffic accountability for types HH transit centers;

(6) In entry-exit switching centers, and in base distribution systems there is usually a high degree of historical accountability to serve the local subscribers.

Delivery Responsibility - The ability of a switching center to deliver traffic can best be seen from a design point of view -

as a series of graded capabilities each requiring more extreme processing operations and extended storage media. Delivery responsibility ranges from:

(1) Active Delivery. In this state all systems within a switch are working and normal switch processing manages the delivery of traffic;

(2) Delivery from recovery in switching complexes which are duplexed, or even multiplexed; delivery responsibility includes the ability of the switch to maintain delivery accountability in the event of loss of one or more duplexed components of the switching center hardware. In this case recovery is usually "instantaneous" with little or no resultant loss in service to the customers;

(3) Delivery from restart. This implies the ability to maintain delivery accountability after the total shutdown, but not destruction, of the active switch processors. The switching system itself is restarted, and all active tables and dynamic queues are refreshed from the Journal, Historical/Reference, Intransit and Ledger storage files maintained in logically and electrically separate media;

(4) The switch is released from delivery responsibility when it has processed the EOM signal for the message and/or received one or more system-level defined acknowledgement procedures from the down-stream station.

Delivery acknowledgement procedures impact on overall message switch processing. Such procedures will be specifically included in the traffic level parameters to be applied to the processing models and analysis of selected switch configurations performance. Delivery acknowledgement ranges from:

(1) Anti-acknowledgement - usually treated in a time threshold in which the absence of a NACK within a certain period of time indicates successful receipt by the down-stream station. This is the least expensive in terms of overall traffic requirements, especially high level trunks;

(2) Acknowledgement to previous transit center. This is the most obvious and universal type of acknowledgement procedure and is performed on either a block by block - for high level trunks - or on a message by message basis for low level tributaries or interswitch trunks not operating on a high level blocked basis;

(3) Receipt acknowledgement to entry center or originator. This type of acknowledgement procedure results in a significant increase in total character traffic over the high level trunks of the system. It is usually associated with network accountability intentions and significantly raises the overall character traffic on the high level lines. Exit centers must bounce back acknowledgement receipt through each transit center back to either the entry center and/or originating terminal station itself.

<u>Critical Processing Factors</u>

There are several critical processing and procedural factors which, when combined in various mixes, account for and support the various levels of system accountability. These are:

(1) Retention (storage) of messages throughout the system at one or more swtiching centers. Retention of messages at given switching centers varies significantly according to the accountability intention of the system (BR-67). Message retention in volatile core storage is maintained until the time of immediate acknowledgement from the down-stream center. At the other extreme, messages are retained in some medium of storage at a given switching center for a period of one, ten and perhaps as many as thirty days before destruction. During this time the message may traverse through a hierarchy of storage media from fast access (for purposes of retrieval) storage to magnetic tape storage or off-line disk packs.

(2) A recording in formatted information records - journal entries - of the events of processing for each message at a given switching center. The journaling or event monitoring of message processing also received a wide range and diversity of processing treatment in the systems studied. At one end of the spectrum (for example, the ARPA IMP nodes, and the HH function processing of the SITA switches) message events are "journaled" in dynamic queues and tables maintained in core. The journal entries for a given message are active only during the time the message is in core - from the time of its receipt from the upstream station until the time of the receipt of acknowledgement from the down-stream station. Thus these event journals are dynamic and provide little if any post-mortem analysis as to the processing and/or disposition of the message after its "time" in the switch. In these particular cases there is no recording of the journal

events on a separate storage media. At the other end of the spectrum, represented by the highly accountable military systems, a carefully defined set of message processing events are generated and recorded on external media (sometimes replicated) for the purpose of post-mortem analysis and tracing of the message history. This includes copies of the message itself. Thus at the one extreme, once a message is delivered and acknowledged it is gone from the switch forever. At the other extreme the message, and its associated audit trails, "stay with the switch" for a designated period of time.

(3) A third factor of switching center accountability for message delivery is the manner and degree to which the system (an individual switching center) can recover from varying degrees of component failure - and still maintain the ability to deliver those messages as yet undelivered and/or maintain audit trail journals of processing activities that occurred up to the time of failure. Here again a wide range of functional capabilities is exhibited in the systems studied. At the simplest end, as the switch goes down all is lost - i.e., by the failed switch. The switching system may be brought back to life from an operational point of view but all traffic for which the system acknowledged a delivery responsibility - from a network point of view - is lost. There are significant variants and gradation of recovery accountability procedures between this rather austere extreme and the opposite end of the spectrum. At the opposite end most military oriented switching complexes are provided with several degrees of graceful degradation and recovery points. This involves the use of duplexed equipment, duplexed (redundant) storage of the several accountability-oriented files and includes periodic ledgering of the active states of the switching processes. Full accountability and processing activity can be restored and maintained for all messages for which a particular switching center has acknowledged (assumed) responsibility for delivery, that is, short of total destruction of the entire switch site.

(4) The final critical factor is the nature of the network accountability procedures and doctrines. This is the degree to which messages are maintained (replicated) at one or more switching centers in the system during the transit time of the message in the system, and in fact for a designated period after the message has been delivered to all intended addressees.

Levels of System Accountability

The nature of accountability processing installed across the several switching centers of a communication system is fundamentally dictated by the needs of accountability management for the system as a whole. Thus accountability intentions are initially defined on a total network and traffic management basis. This in turn dictates the nature of accountability processing and computer resources required at each of the switching centers which comprise the system.

Network level accountability doctrines may be classified into 5 or 6 levels of accountability performance by a communications system. A brief review of these levels is essential to the ensuing analysis of the requirements for processing at a given switch.

It should be stressed that the concept developed in the following should not be seen as a monolithic to a communications system. Rather, accountability applies in levels and is assumed to be associated with various kinds of traffic managed by the system. Several existing systems, e.g., AUTODIN, SITA, and SAMSON, incorporate, or are designed to incorporate, several levels of accountability for the various classes of traffic.

The classes currently are roughly defined by various combinations of message priority, security, and other classifications in the absence of a specific accountability designator assigned by the message source. The assignment of accountability message designators is a concept that is only beginning to appear in the design specifications of systems that will probably be built in the near-term future.

User Level Accountability

As a network intention, user level accountability involves no accountability whatsoever and therefore poses a very minimum requirement for computer resources and processing. An individual switch would indeed contain a message in some storage level until it is ready for transmission and would indeed retain a copy of that message at least until the receipt was acknowledged by the immediate down-stream station. At a minimum, there might be some threshold set; such as the number of automatic retransmissions from one center to the next. Partial or full destruction of a switch would result in loss of undelivered traffic and the sender would, according to his own procedures, be forced to retransmit the message. Thus all accountability is a matter between the sender and receiver and transmission and receipt acknowledgement are themselves message level transmissions using systems resources.

Network Accountability

This form of accountability is frequently called entry-exit accountability. In essence a network accepts a message at an entry center which maintains full accountability for the delivery of that message and the retransmission thereof until directly acknowledged by the switching center that is the last center of forwarding to the final destination unit. The exit center acknowledges via a system level message directly back to the entry center that the message has been successfully delivered. The entry center may in turn acknowledge the sender via a system level message. Intermediate high-level S/F centers maintain accountability only for active messages; the message to be released when acknowledged by the next down-stream station. The up-stream transit center may maintain historical audit trail records of the message for purposes of tracking lost, undelivered or misrouted messages by tracing their events.

The rapid transfer tandem centers maintain minimum historical accountability, and the exit Regional center acknowledge the entry Regional center of the receipt and forwarding of the message. The implications are that for pure high-level transit forwarding centers little off-line storage is required for message accountability. Message retention for long term retrieval accountability is a decision factor applied to the entry and exit center as to which will receive the extended treatment.

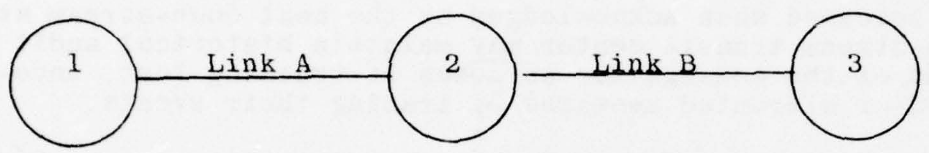
Node by Node Accountability

In a further step to assure deliverability of message traffic (particularly in military systems designed for a hostile environment) node-by-node accountability is achieved through full historical and traffic accountability procedures installed at each and every node of the path traversed by the message through the system. It is applied to all messages designated to receive this type of accountability management. The point is that the critical accountability processes of electrically and logically separate independent storage of both message contents and event journals are maintained fully at each node through which the message passes. The accountability for message delivery at a given node is released when that node receives acknowledgement from the down-stream station. Full accountability procedures are invoked at each center for traffic passing through that center. This provides a much more highly reliable system than that of network accountability, but of course increases processing and storage requirements at each switching center.

Link by Link Accountability

This particular form of network accountability is a logical extension of the node by node accountability. A link bracketed by the nodes (switching centers) at each end of the link is considered as the accountable unit. Therefore accountability is maintained at each switching center at both ends of the link comprising a fully accountable unit.

The message moves from link A to link B. These are bracketed by nodes 1 and 2 at the ends of link A, and nodes 2 and 3 at the ends of link B.



What occurs is that full accountability of a message is maintained in link A at both nodes 1 and 2, while at another time accountability is maintained at link B, at nodes 2 and 3. The net effect is to increase the overall storage and processing requirement of each of the nodes in the entire system. The effect of this, of course, is significantly increased system reliability and performance in the event of nodal or trunk failure. The destruction of any single node or any single link will not result in the loss of the message nor of the history of its activities.

Path Accountability

The ultimate accountability - and most costly - is one in which each switching center traversed in the path of the message through the system maintains full and complete message and history accountability for messages designated for this accountability treatment. Release of either message or historical accountability would be performed upon complete acknowledgement of the message, and of its delivery, to all addressed receivers.

Accountability Requirements for a Switching Center

The previous section discussed accountability intentions on a network-wide basis. For the purposes for input to the CPS architectural design however these must be translated to their impact on processing requirements, storage hierarchies and configuration organization to an individual switching complex. In this section, three levels of accountability intentions for application to a switch will be postulated.

It should be understood that the orientation of accountability processing is towards individual categories of message accountability. This is distinct from the total accountability processing performed by a switching center. Indeed, a switch may be dedicated to handling only one class of accountable traffic, or the switch may be designed to handle several classes of accountable traffic. For practical purposes, a switch would have to be configured with resource expandability to the highest level of accountable traffic to be processed.

Minimum Switch Accountability

The characteristics and requirements for minimum accountability processing can be summarized by the following:

(1) The delivery responsibility for the switch which is usually restricted to active responsibility. That is as long as the switch is up and running it will deliver traffic for which it has acknowledged receipt and forwarding responsibility. (Clearly traffic which has been received by the switch but which has not been acknowledged to the up-stream station, by the exercising of some acknowledgement protocol, is not lost in the system since the switch has not accepted delivery responsibility - prior to failure.)

(2) The intransit storage medium for messages - in whatever envelope is being processed - is usually volatile core memory. In addition, the system configuration may be a simplex one for absolute minimum cost. A slight variation on this would be a separately powered core memory shared by duplex processors to achieve redundancy for active delivery responsibility.

(3) There is usually virtually no traffic or historical accounting - save for the active accountability maintained in the core tables, delivery queues and channel queues.

(4) The switch assumes release for active delivery responsibility according to any one of several acknowledgement protocols that may be established on a network basis for high level traffic. These range from positive block acknowledgement from the down-stream center, to negative logic NACK procedures. Positive acknowments are directed to the immediate upstream switch for acknowledgement of immediately received block. (Network level exit/entry acknowledgement is treated simply as a high-level traffic message by the high-

level transit centers, or in some cases they are piggy-backed onto message packets otherwise traveling to the desired entry center.)

(5) With minimum switch accountability the following functions are not present:

(a) there is no capability for delivery or retransmission after switch failure and/or re-start;

(b) there is no retrieval after delivery, there is no capability for user requested retransmission;

(c) there is no message event journaling on separate storage media and therefore there is no capability for post-mortem tracing of message events to determine status, disposition, routing, etc.;

(d) there may, or may not be a rudimently form of message processing statistics - in terms of message (packet) counts, character counts, statistics on acknowledgement, negative acknowledgement, etc.

(6) With minimum accountability there is frequently included in the line protocols ARQ (automatic repeat request). This may be incorporated as a result of a reply to a NACK from the down-stream station or may be a result of not having received a positive ACK within some system defined time threshold.

(7) Minimum accountability switching configurations then require:

(a) no external (storage media) intransit file (except as an economy measure).

(b) no external storage media for reference or journal file;

(c) no external storage media for the activity ledger file;

The switch architectural requirements, then call for:

(1) A small number (5 to 20) of medium to ultra-high speed channels with a range from 2400, 4800, up to 300K BPS channels.

(2) There is practically no peripheral storage requirements and a minimum peripheral output/input and devices such as floppy disc or cartridge tape drive for program loading. A keyboard visual display unit could be used for system maintenance, system/network configuration control, real time status monitoring, and table and routing changes.

(3) There is a minimum requirement for message format, speed, code, header processing of the rigidly but simply formatted envelope headers. This is a simple process and routing analysis (since all packetized messages <u>should never have more than one address per packet</u>) is also a simplified procedure.

(4) Most of the requirements are for dynamic core buffer management of message segments. Much of these dynamics can be reduced to a minimum through application statistics of channel input/output activity. Proportionate buffering can be assigned quasi permanently to balance the input/output requirements of high level trunks connected to the switch.

Intermediate Switch Accountability

An intermediate level of accountability can be conceived as one, or more steps in several directions of maintaining delivery capability from some level of switching center failure. There are several degrees of variability in the spectrum of accountability. These can be seen as:

(1) Degree of replication of processor and storage system components.

(2) Historical accountability

(3) Traffic accountability

Historical Accountability

A reasonable first step in the direction of increased accountability would be to provide a historical accountability for messages processed by the switch. This would be accomplished by the recording of predetermined processing events for each message, written to an on-line storage device. This could be applied in a reasonable manner to the processing of high-level packet-type messages in a forwarding transit center. A trace of the arrival and transmission of messages would be accounted for. This would add to the processing load for each message-packet and would require a high speed channel to the storage device. Note that only message events would be recorded not necessarily the messages themselves - for historical accountability. In this construct, accountability is otherwise at a minimum and the Journal provides post-mortem event analysis. It would not be used to provide delivery responsibility in the event of recovery.

Recovery Accountability

The next logical construct of increased switch accountability would be to provide delivery responsibility in the event of failure of one or both (if duplexed) of the active message processing systems. Delivery responsibility would be provided after such failure by the inclusion of three processing procedures and additional on-line storage. This includes:

(1) The recording of message events in the Journal file.

(2) The recording of messages for which the switch maintains delivery responsibility on an Intransit Storage file on some Random Access Storage (RAS) device. The intransit storage area would be used only for messages with an active delivery responsibility. Space is released after the switch has fulfilled delivery.

(3) Some measure of status ledgering would be required to maintain the storage structure of the messages (message blocks) maintained on intransit storage file. These process ledgers might be written as part of a directory on the intransit file itself, or on Journal storage.

This scheme would provide an off-line form of recovery - not the instantaneous recovery frequently required for redundant switching systems. All three of these processing components would be required to achieve this level of accountability - the journal event, separate instransit storage, and ledgering of the intransit storage directory. There is no traffic accountability for either retrieval or retransmission after the switch has released itself of delivery responsibility of any given message.

The additional processing on a per message basis - as compared to the minimum accountability of the previous section includes the following:

(1) the writing of the message events to the journal file.

(2) the block gathering writing of message segments to the intransit storage file, and the read of intransit storage file during the output processing of these messages.

(3) the writing of the intransit directory ledgers.

Traffic Accountability

Traffic accountability could be added as an independent construct by the addition of the History/Reference file. In this scheme messages (message blocks) are recorded as received during the input processing portion of the processing cycle on a separate recording medium. If message blocks only were recorded, then this file would be used only for retrieval/retransmission upon request by searching (passing) the entire file to gather the separately recorded blocks of the requested message. This file by itself could not be used for recovery in the event of failure since it would contain no journaled events of what messages were delivered, their status, etc.

This traffic accountability scheme would be added independently of the other accountability processes described above, or could be added in any desired mix of them. The additional processing resources required are:

(1) the writing of message blocks as received in the host processor core;

(2) an appropriate I/O channel to the on-line storage device;

(3) in addition, retrieval, retransmission functions would have to be provided for tape searching, message reconstruction, and delivery and output processing.

Maximum Switch Accountability

Maximum accountability for a given switching center involves an extensive treatment of both historical and traffic accountability. These procedures are called for in highly strategic and tactical military environments. The environment is characterized by expected hostile activities in which there is a potentially high threat of message compromise, or destruction of one or more elements (lines and switching centers) of the communications system. Alternatively, the very procedures of traffic accountability support other functions such as message retrieval which provide for user oriented dissemination of message information.

Full message accountability procedures require extensive computer configuration resources - in comparison to minimum accountability processing - and adds significantly to the overall processing requirement on a message-by-message basis by each switching center. Full accountability procedures add to the total storage requirement and add to the transient delay time of the message in the switching center.

Maximum accountability traffic is usually installed at entry and exit centers of long haul communications systems. In contrast to HIGH-HIGH trunk transmission, full accountability procedures are usually asociated with local level transmissions, LOW-HIGH, and HIGH-LOW traffic.

Typical characteristics and requirements for maximum accountability can be summarized by the following:

(1) Through a high degree of component redundancy in both active processors, and on-line and off-line peripheral storage equipment, the switching center is able to maintain not only an active delivery responsibility but also delivery responsibility in the event of the loss of one, or several active and storage components. Delivery responsibility from varying degrees of recovery standpoints is a simple cost function. The greater the degree of component redundancy the greater the capability for maintenance of delivery responsibility both active and in restoration.

(2) The storage medium of active delivery responsibility - intransit storage - is typically duplexed in on-line storage media. The redundant recording of messages intransit for delivery provides the typical two-level reliability backup.

(3) In addition to the active delivery storage media (intransit storage), message segments are also written to electrically and logically independent Reference files for extended backup required under full system recovery procedures.

These reference files are typically tape files in which message segments are written serially as received during input processing, and blocked according to system design parameters. Each and every message, or message segment, received during input processing is written to the Reference file. This is usually a highly complex formatted file containing segments of all messages received - acknowledged, or not acknowledged, full messages, partial messages, stragglers, etc.

(4) A significant part of full accountability processing involves the recording of activity - status ledgers onto on-line storage units. This involves the dynamic tables maintained in core including the output queues, channel queues and other processing status indicators. The activity ledgers recorded on on-line equipment are an absolute necessity for recovery from a failed

duplex host computer and are also used in total system restart for balancing the delivery responsibility of messages in the Intransit and Reference files.

(5) The switch assumes release from active delivery responsibility of a message - that is its release from intransit storage - when all deliveries of the message have been certified acknowledged by the appropriate network level accountability/acknowledgement procedures. After release from delivery responsibility, the message space on the instransit store is freed - through use of dynamic space availability tables for re-use by incoming messages.

(6) Complete historical accountability is provided by the recording of processing events associated with each and every message entering and leaving the switching center. These events are time-stamped, formatted, and written to logically and electrically separate on-line storage devices. Frequently the Journal file is combined logically with the History/Reference file. The Journal records can be used to provide and prepare time oriented history of any given message events on demand.

The impact on system architectural requirements for full message accountability involves:

(1) Redundant on-line storage devices comprised of disc, drums, bubble memories, and tapes. These are usually structured in a hierarchy of responsiveness to the demands for historical or traffic recall.

(2) Access ports to the on-line storage devices call for DMA channels of high or variable capacity and easily prepared software device handling modules.

(3) Accountability procedures frequently require backup storage of message traffic for some period of time ranging from one day to thirty days. This requires an adequate supply of physical storage units (movable disc packs, tapes, tape cartridges) and adequate space and supervisor personnel for management.

(4) With on-line separated media storage for intransit traffic, software processing tools for space availability tables require significant bit processing and bit-imbedded in table-processing logical operations. These are in addition to the dynamic core buffer management for message segments required for even minimum accountability procedures.

2.1 Specific M/S System Parameters

For the purposes of comparison, three types of M/S system applications will be considered. These are:

(1) A 40 port Packet Switch configured in a network as shown in Figure 2.1-1.

(2) A 170 port Store and Forward M/S configured in a network as shown in Figure 2.1-2.

(3) A 380 port Base Distribution M/S configured in a network as shown in Figure 2.1-3.

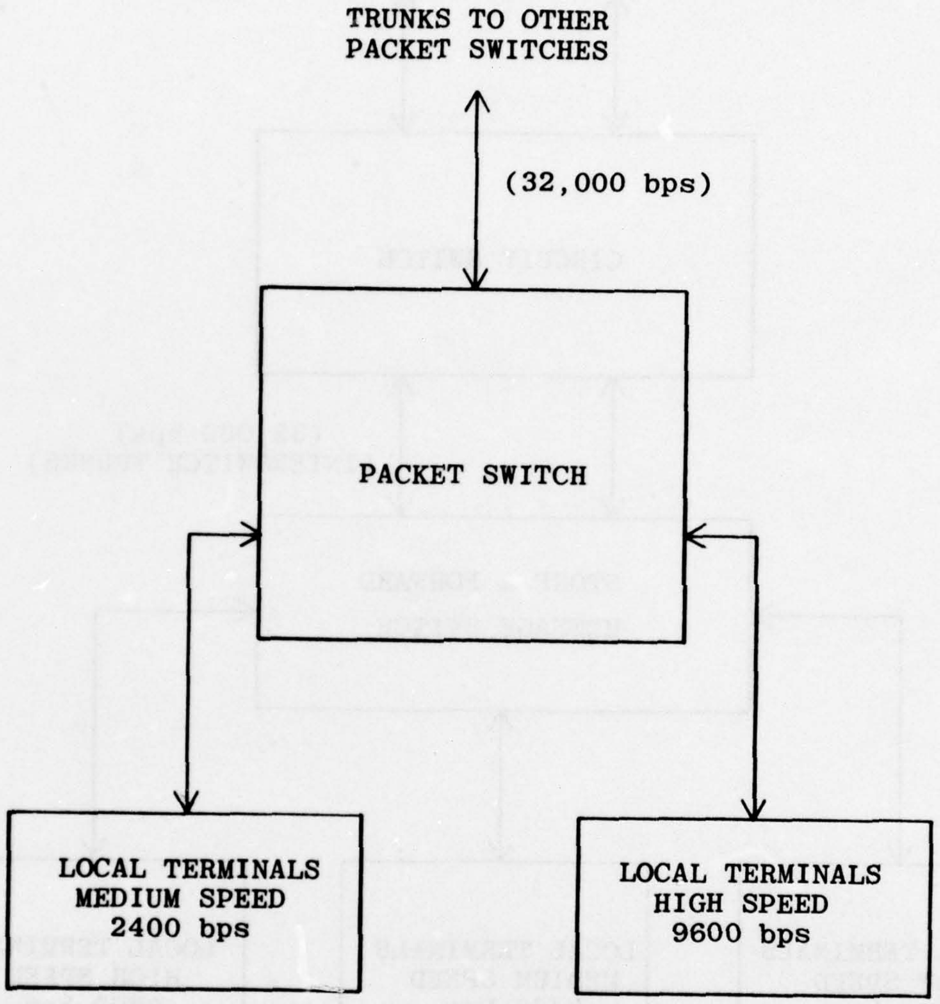
Tables 2.1-1 and -2 describe the salient functions, characteristics, and specific parameter values for each switching system.

Message Distribution

The message distribution percentages given for narrative and data traffic by line block and by message, indicate current or expected (near term 1 to 2 years) distributions. The best estimates for the TRI-TAC system, S/F Switch, represent probably a balancing out of processing requirements between narrative and data traffic. It should be remembered that narrative traffic requires more processing per message and more complex processing per message in contrast to data traffic due to the factors of address multiplicity, format and code conversion, and other addressing requirements not usually contained in data pattern messages. For those switches which process packetized messages the distinction between narrative and data traffic poses no special requirements on the processor, since the type of message is transparent to the packet oriented processing.

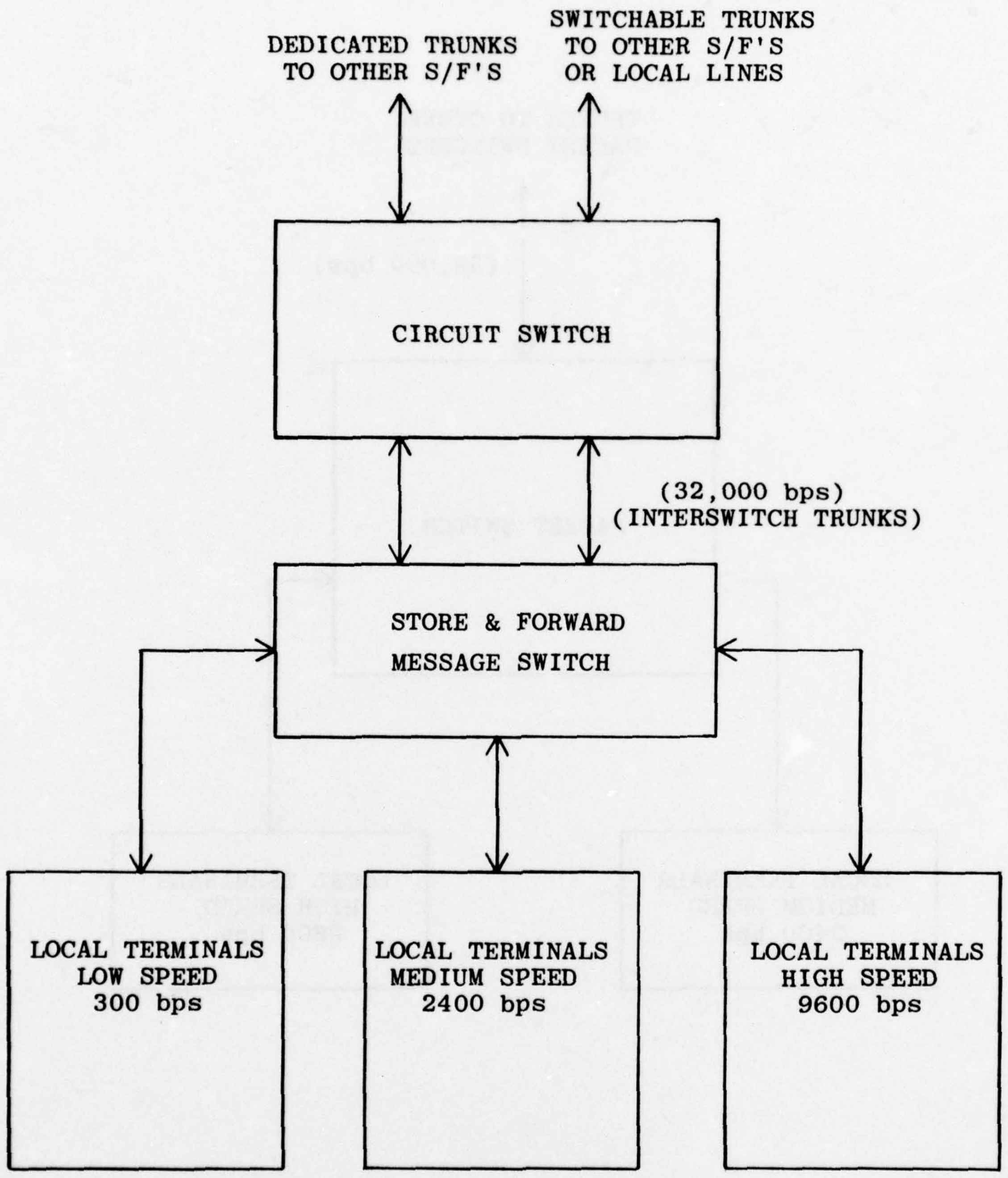
Message Length

These figures represent an approximate average length of messages across the systems. In all cases maximum length of allowable messages may be 10 to as high as 100 times the indicated average length. In general the message lengths range from approximately 1.5K characters at the low end, to approximately 2-3K characters at the medium, and up to 30K characters at the high end of the spectrum. The significant impact of the variations in the average message length would be in the requirements for intransit storage at switching centers for which full accountability procedures are implemented. The packet oriented processing switches see only small packets and therefore total message length has little impact on processor requirements for storage, not withstanding total throughput requirements.


TRUNKS TO OTHER PACKET SWITCHES
(32,000 bps)
PACKET SWITCH
LOCAL TERMINALS MEDIUM SPEED 2400 bps
LOCAL TERMINALS HIGH SPEED 9600 bps

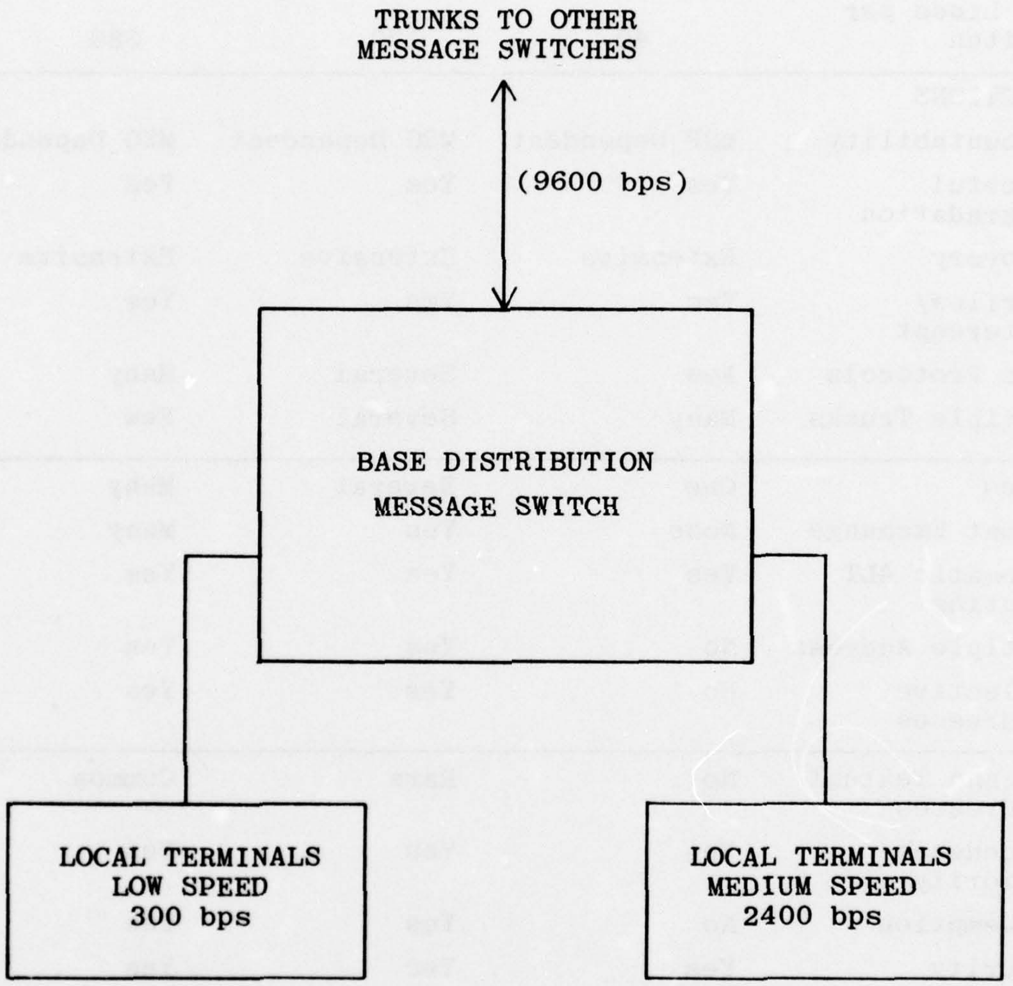
PACKET SWITCH UTILIZATION

FIGURE 2.1-1



STORE AND FORWARD M/S UTILIZATION

FIGURE 2.1-2


TRUNKS TO OTHER MESSAGE SWITCHES
(9600 bps)
BASE DISTRIBUTION MESSAGE SWITCH
LOCAL TERMINALS LOW SPEED 300 bps
LOCAL TERMINALS MEDIUM SPEED 2400 bps

BASE DISTRIBUTION M/S UTILIZATION

FIGURE 2.1-3

TABLE 2.1-1
CLASSES OF M/S SYSTEMS

Function or Characteristic	Packet Switch	S/F Switch	Base Distribution
Intransit Store (Line Blocks)	20K	60K	120K
No. Lines per Switch	40	170	380
FUNCTIONS			
Accountability	MSF Dependent	MSG Dependent	MSG Dependent
Graceful Degradation	Yes	Yes	Yes
Recovery	Extensive	Extensive	Extensive
Overflow/ Intercept	Yes	Yes	Yes
Line Protocols	Few	Several	Many
Multiple Trunks	Many	Several	Few
Codes	One	Several	Many
Format Exchange	None	Yes	Many
Automatic ALT Routing	Yes	Yes	Yes
Multiple Address	No	Yes	Yes
Collective Addresses	No	Yes	Yes
PLA and Textual Addresses	No	Rare	Common
Precedence/ Priority	Yes	Yes	Yes
Pre-emption	No	Yes	Yes
Security	Yes	Yes	Yes
Privacy	Minor	Minor	Major
Header Validation	Rudimentary	Complex	Very Complex
Exception Processing	Minimum	Complex	Extensive
Traffic Management	Minimum	Extensive	Extensive
Retrieval/Trace	Yes	Yes	Complex

PLA = Plain Language Address
MSG = Message

TABLE 2.1-2

TRAFFIC CHARACTERISTICS OF SYSTEMS

TRAFFIC CHARACTERISTICS	BASE DISTRIBUTION	S/F SWITCH	PACKET SWITCH
Message Distribution			
% By Message (N/D)	40/60%	30/70%	N/A
Multiplicity	4	2	1
Msg. Processing Delay			
Second/Msg.	2	.85.2	< .06
Throughput			
Sustained Char/Sec	95K	36K	20K
MSG Arrival/Sec	1	5	20 (Packets)
Traffic by Line Speed			
% Low (up to 600 BAUD)	5%	5%	0%
% Medium (1200,2400,4800)	85%	10%	5%
% High (9600)	10%	15%	10%
% Very High (32K)	0%	70%	85%

N/A	= Not Applicable
N/D	= Narrative/Data
MSG	= Message
CHAR	= Characters
AVG	= Average

Multiplicity

Specifications for switching systems, and associated software procedures allow message address multiplicity to range as high as 250 (for the SAMSON system). This is seen as a logical complexity in contrast to a processing requirement. The average multiplicity of most messages is and will continue to be low - in the range of 1 to 3 or 4 (which is high as an average multiplicity). As expected, those systems oriented towards base distribution functions indicate higher multiplicity factors in comparison to the switching centers for long-haul communications.

Message Processing Delay

For the systems represented, message processing delay (seconds per message) required for processing at a M/S site reflects the dichotomy of packet processing and fully accountable processing. Message processing delay for dedicated packet processing operations is seen to be in the range from 10 to 50 milliseconds per packet. In contrast, delay at rated throughput for accountable systems, seems to be still at the 1-3 second range.

Throughput

In the ensuing modeling analysis, two important characteristics which will be parameterized and tested for impact on processing resources, are the sustained character-per-second throughput rate and, the rate of message arrivals per second. Both are important since each parameter contributes its own specific requirements for CPS processing and performance features. An increase in the number of message arrivals (per second), regardless of length, significantly increases the total processing required, since the complexity of the message is generally in the header processing and not in the text processing. On the other hand, total throughput through any limited resource "pipe" is bounded by the "pipe" capacity.

Retrievals

All the existing systems showed a relatively low rate of retrievals for message retransmission. However, retrieval requests require considerable processing and it is expected that the logic of retrieval requests will become increasingly complex with systems for the intermediate term period (5-8 years). Currently the values for retrieval requests are ranged at around (or under) 5 per hour, however, it is not unreasonable to anticipate that this could jump sharply to perhaps 25-50, or even 100 per hour, for base distribution systems with enhanced message/data base processing retrieval by text-oriented queries.

Traffic Distribution by Line Speed

The distribution of traffic (in terms of characters per second) over the range of various line speeds is an important factor effecting the CPS processing requirements. The system studied suggest an era of gradual growth from medium, and high speed-line to the very and ultra high speed-line of 50K to 300K BAUD. The most significant result of the study is that the wide range of line speeds will continue to exist and therefore pose these requirements on the CPS processing. Switches designed for packet oriented processing require the very high to ultra high speed lines. Entry-exit switches require a wider range of line interfaces from the medium to ultra high speeds. Base distribution systems directly interfacing low speed terminals will of course require appropriately low speed line adapting units. It is expected that the replacement of the enormous capital investment in low speed teletypes, stilll used as a primary message entry station, will be a long time in coming. This is not withstanding the tremendous impact that relatively cheap CRT keyboard stations utilizing integrated circuitry are having on the marketplace. (The net effect of buffered terminals is to increase the line speed.)

2.1.1 Packet Switch Parameters

Calculations for the Packet Switch (P/S) system are based on the characteristics of Table 2.1-2 and the following assumptions:

(1) Three line speeds are assumed.

(a) Very High speed Line (VHL)

Operates at 32000 bps

(b) High Speed Line (HSL)

Operates at 9600 bps

(c) Medium Speed Line (MSL)

Operates at 2400 bps

(2) Average message length = 500 characters.

(3) Average packet length = 500 characters.

(4) Header length (fixed format) = 50 characters.

(5) No multiplicity - one R.I. per message.

(6) All messages use same code and formats.

(7) Line block length = 50 characters.

(8) Message Distribution (BH - originating and terminating).

Percent by Line Speed	Total Messages	Traffic	Terminals
MSL 5%	7200	7E	17
HSL 10%	14400	3E	10
VHL 85%	122400	4.25E	13
TOTAL 100%	144000	14.25E	40

(9) Message Distribution by Line Speed

Line Speed	Originated Messages	Terminated by Line Speed VHL	HSL	MSL
VHL	61200	52020	6120	3060
HSL	7200	6120	720	360
MSL	3600	3060	360	180
TOTALS	72000	61200	7200	3600

(10) All messages are acknowledged by the receiving station (switch or terminal) on a packet by packet basis.

(11) Channel coordination - a sync control character is received with each packet. Also a sync control character is sent with each message.

(12) Average Holding Time ($\overline{HT}$) per message by line speed:

VHL $\overline{HT}$ = .125 sec

HSL $\overline{HT}$ = .75 sec

MSL $\overline{HT}$ = 3.50 sec

(13) Accountability

Extry-Exit Accountability

(a) Entry Traffic - Maximum accountability (network) assumes complete responsibility for delivery of message. Maintains messages for long period for retrieval and retransmission.

(b) Exit Traffic - Historical/Recovery accountability. Provides historical record of all messages which exit the network. Maintains recording of messages until delivery to receiving station is acknowledged.

(c) Transit Traffic - Minimum accountability. Releases messages from intransit storage when next P/S has acknowledged receipt. Maintains delivery and channel queues. Keeps traffic statistics, i.e., message count

character count

statistics on acknowledgement

traffic/trunk group

trunk and line status

etc.

(d) Entry and Exit Traffic in the same switch, i.e., between locally homed terminals, is treated as Entry traffic with maximum accountability.

(14) All MSL and HSL lines are local terminals.

(15) No plain language or textual addressing.

(16) 5% of all message blocks (packets) require retransmission.

(17) Average of 6 characters per R.I.

<u>Calculations</u>

(1) Local-to-Local

MSL to HSL	=	360
MSL to MSL	=	180
HSL to HSL	=	720
HSL to MSL	=	360
TOTAL	=	1620 messages/BH

(2) Local-to-Trunk

MSL to VHL	=	3060
HSL to VHL	=	6120
TOTAL	=	9180 messages/BH

(3) Trunk-to-Local

VHL to MSL	=	3060
VHL to HSL	=	6120
TOTAL	=	9180 messages/BH

(4) Trunk-to-Trunk

VHL to VHL = 52020 messages/BH

(5) Total number of headers:

(a) Validated = 72000/BH

(b) Generated = 72000/BH

(6) Total number of R.I.'s:

(a) Validated = 72000/BH

(b) Generated = 52020/BH

(7) Total network entry messages/BH = 9180
(maximum accountability)

(8) Total network exit messages/BH = 9180
(historical/recovery accountability)

(9) Total entry/exit messages/BH = 1620
(maximum accountability)

(10) Total messages receiving maximum accountability
per BH = 10800
per sec. = 3

(11) Traffic Statistics (BH)

Incoming messages (packets)	=	72,000
Incoming characters	=	36,000,000
Incoming packets requiring retransmission	=	3,600
Outgoing messages (packets)	=	72,000
Outgoing characters	=	36,000,000
Outgoing packets requiring retransmission	=	3,600
TOTAL INCOMING PACKETS	=	75,600/BH
Incoming characters	=	37,800,000/BH
		10,500/sec
TOTAL OUTGOING PACKETS	=	75,600/BH
Outgoing characters	=	37,800,000/BH
		10,500/sec

2.1.2 S/F Switch Parameters

The calculations concerning the S/F switch are based on the values stipulated in Table 2.1-2 and on the following assumptions:

(1) 24% of the terminals of the S/F are terminated on a collocated C/S. (A total of 40 terminals are terminated on the C/S.)

(2) 15 terminals are permanently patched through the C/S as dedicated trunks in trunk groups to other M/S systems.

(3) 25 terminals, terminated on the C/S, are switchable to other outgoing trunk groups or to local subscribers homed on the S/F through the C/S.

(4) All lines to the C/S operate at 32 Kbs.

(5) Three line types are used whose characteristics are as follows:

(a) Type 1: Low Speed Line (LSL) (300 bps)

Operates in AUTODIN modes II, IV and V. Uses ITA #2 code in JANAP-128 or ACP-127 formats.

(b) Type 2: Medium Speed Line (MSL) (2400 bps)

Operates in AUTODIN modes II, IV and V. Uses ASCII code and JANAP-128 or ACP-127 formats.

(c) Type 3: High Speed (HSL) or Very High (VH) speed line (9600 and 32000 bps).

Operates in AUTODIN modes I and III. Uses ASCII code and JANAP-128 format.

Therefore: 2 codes, ITA #2 and ASCII, are used; 2 formats, JANAP-128 and ACP-127 are used.

(6) Channel Coordination

Type 1 Line

Data receive - recognize a steady mark (stop) signal as the idle pattern and a transition from mark to space as the start of the character frame.

Data transmit - two methods used;

(1) transmit data rhythmically with no external character step.

(2) transmit each character upon the receipt of a character step (release) pulse and maintain a steady mark condition upon completion of a character until the receipt of the next step pulse.

Assume that 1/2 of the Type 1 lines transmit with method 2.

Type 2 Line

Same as Type 1 line as far as coordination procedures are concerned.

Assume that 1/2 of the Type 2 lines transmit with method 2.

Type 3 Line

Data receive - data control characters used on a 84 character block basis. Receives idle pattern for synchronization, verification and channel checking purposes.

Data transmit - data control characters transmitted with each 84 character block. During idle periods, transmits the idle pattern.

(7) Language Media Format (LMF)

Three types of LMF's are assumed:

(1) card format

(2) paper tape format

(3) magnetic tape format

Any LMF may be used with any message.

Assume an even distribution of LMF type for all line type messages.

(8) Average Holding Time ($\overline{HT}$) per message (2400 characters) is as follows for each line type:

(a) Very High speed, Dedicated trunk, (VHD)

$\overline{HT}$ = 1 second. This includes .6 seconds message transmission time and .4 seconds "dead" time.

(b) Very High speed, Switchable trunk, (VHS)

$\overline{HT}$ = 3 seconds. This includes .6 seconds message transmission time and 2.4 seconds tandem signaling time.

(c) High Speed Line (HSL)

$\overline{HT}$ = 5 seconds. This includes approximately 2.5 seconds for signaling, line coordination or "dead" time.

(d) Medium Speed Line (MSL)

$\overline{HT}$ = 10 seconds. This includes 2 seconds for signaling, line coordination or "dead" time.

(e) Low Speed Line (LSL)

$\overline{HT}$ = 100 seconds. This includes 10 seconds for signaling, line coordination or "dead" time.

(9) Maximum traffic for 170 terminations S/F during the BH is 120E.

(10) Average of 5 message arrivals per second during the BH.

(11) Message Accountability Distribution

Minimum Acc.	20%
Intermediate Acc.	20%
Historical/Recovery Acc.	20%
Historical-Recovery-Traffic Acc.	20%
Maximum Acc.	20%

(12) Accountability Functions

(a) Minimum Accountability

Buffer storage for active processing

Maintains delivery queues

Maintains channel queues

Releases messages (removes from intransit storage) upon receipt of acknowledge by receiving equipment.

Traffic statistics are kept, i.e.,

message counts

character counts

statistics on acknowledgements (ACK's, NACK's)

(b) Intermediate Accountability (Historical)

provides, in addition to minimum accountability facilities; historical records of predetermined processing events written to an on-line storage device. A journal of message events is kept.

(c) Historical/Recovery Accountability (H/R)

in addition to the above, it provides recording of messages on an Intransit Storage file. Some measure of status ledgering is performed to maintain the storage structure of the messages (or blocks).

(d) Historical/Recovery/Traffic Accountability (H/R/T)

in addition to the above, it provides recording of message blocks as received on separate recording medium. This allows retrieval and retransmission functions to be performed.

(e) Maximum Accountability

All functions of above accountability procedures plus added redundancy and storage capabilities. Reference files are kept. Complete status ledgers redundantly stored.

(13) No plain language or textual addressing.

(14) Message Distribution by Line Speeds

VHD	42.5%
VHS	27.5%
HSL	15%
MSL	10%
LSL	5%

(15) All messages are transmitted in fixed line blocks of 84 characters each.

(16) Headers are of two types; single RI per header in which the entire header is contained in a single line block; and multiple RI's per header in which the header is contained in 2 blocks. An average of 1/2 of input messages contain multiple header (or 2 line block headers).

(17) Header Validation includes validation of all fields up to the "start of routing" indicator. Rigid validation procedures used on all messages.

(18) Routing Validation includes checking the originating station identifier and serial number, checking each RI against the RI Table, and formatting the outgoing RI's on messages to other M/S's.

(19) Block Retransmission required for 5% of all trunk messages (incoming and outgoing).

Calculations

(1) 5 messages/sec. terminate on the S/F equals 18000 messages per BH.

(2) With an average multiplicity of 2, 36000 messages are originated by the S/F per BH.

(3) Total BH messages = 54000.

(4) Total messages by line speed:

VHD	(.425) (54000)	= 22950
VHS	(.275) (54000)	= 14850

HSL	(.15) (54000)	=	8100
MSL	(.10) (54000)	=	5400
LSL	(.05) (54000)	=	2700

(5) Distribution of messages terminated by lines and trunks with respect to originating line speed:

Originating Line Speed	Terminating Line Speed				
	VHD	VHS	HSL	MSL	LSL
VHD	6502	4208	2295	1530	765
VHS	4208	2722	1485	990	495
HSL	2295	1485	810	540	270
MSL	1530	990	540	360	180
LSL	765	495	270	180	90
TOTALS	15300	9900	5400	3600	1800

(6) Codes and Formats

	Code	Format
VHD	ASCII	JANAP-128 (All)
VHS	ASCII	JANAP-128 (All)
HSL	ASCII	JANAP-128 (All)
MSL	ASCII	JANAP-128 (1/2), ACP-127 (1/2)
LSL	ITA #2	JANAP-128 (1/2), ACP-127 (1/2)

(7) Code Conversions

ASCII to ITA #2 = 1710 messages
ITA #2 to ASCII = 1710 messages

TOTAL CODE CONVERSIONS = 3420 messages/BH

TOTAL CHARACTERS CONVERTED = 8,208,000/BH
2,280/sec

Format Conversions

JANAP-128 to ACP-127	=	2700 messages
ACP-127 to JANAP-128	=	4050 messages

Language Media Format Conversion

paper tape to card	4000
paper tape to magnetic tape	4000
card to paper tape	4000
card to magnetic tape	4000
magnetic tape to paper tape	4000
magnetic tape to card	4000
TOTAL LMF CONVERSIONS	24000 messages/BH
TOTAL CHARACTERS	57,600,000/BH
	16,000/sec

(8) Line Assignments and Traffic

VHD	15 Terminations	6.375 E
VHS	25 Terminations	12.375 E
HSL	19 Terminations	11.25 E
MSL	20 Terminations	15 E
LSL	91 Terminations	75 E
TOTALS	170 Terminations	120 Erlangs

(9) Messages per Termination (originating and terminating) per BH by line speed type:

Line Speed Type	Messages per Line per BH	Characters per Line per BH
VHD	1530	3,672,000
VHS	594	1,425,600

HSL	426.3	1,023,600
MSL	270	648,000
LSL	29.67	71,208

(10) Header Validations and Generations

Total incoming messages per BH = 18000

Total Headers = 18000

Header Blocks

One-half of all headers consist of 2, 84 character blocks

Total header blocks = 27,000

Total header characters = 2,268,000

(incoming per BH)

Total outgoing messages per BH = 36,000

Total header blocks to be assembled for outgoing messages = 54,000/BH

Total header characters (outgoing) = 4,536,000/BH

(11) RI Validation and Generations

Total incoming messages per BH	= 18,000
Total single RI's per BH	= 9,000
Total double RI's per BH	= 18,000
Total incoming RI's per BH	= 27,000

Outgoing RI's on trunk messages only.

Total outgoing trunk messages	= 25,200
Total with single RI's	= 12,600
Total with 2 RI's	= 25,200
Total RI's	= 37,800

(12) Accountability

Total number of BH incoming messages treated with:

Minimum Accountability	=	3,600
Intermediate Accountability	=	3,600
Historical/Recovery Accountability	=	3,600
Historical/Recovery/Traffic Accountability	=	3,600
Maximum Accountability	=	3,600

(13) Traffic Statistics (BH)

Incoming messages	=	18,000
Incoming characters	=	43,200,000
Incoming Blocks (84 characters)	=	514,286
Requests for retransmission (incoming) (blocks)	=	25,714
Outgoing messages	=	36,000
Outgoing characters	=	86,400,000
Outgoing blocks (84 characters)	=	1,028,572
Requests for retransmission (outgoing) (blocks)	=	51,428

Total incoming blocks including retransmitted blocks	=	540,000/BH
Total incoming characters	=	45,360,000/BH
		12,600/sec
Total outgoing blocks including retransmitted blocks	=	1,080,000/BH
Total outgoing characters including retransmitted characters	=	90,720,000/BH
		25,200/sec

2.1.3 Base Distribution Switch Parameters

The calculations concerning the Base Distribution Switch (BDS) are based on the characteristics of Table 2.1-2 and the following assumptions:

(1) Average narrative message length equals 1800 characters.

(2) Average data message length equals 30,000 characters.

(3) 40% of traffic is narrative and 60% is data.

(4) Average message length is:

(.4)(1800) + (.6)(30,000) = 18720 characters

(5) Average multiplicity is 4

for each message into the switch, 4 messages are generated by the switch.
18720 (characters in) + 74880 (characters out) = 93600 characters throughput per message.

(6) Average message arrival per second = 1.

(7) Sustained characters/second throughput = 93,600.

(8) Line Types

Three line types are used whose characteristics are as follows:

(a) Type 1: <u>L</u>ow <u>S</u>peed <u>L</u>ine (LSL) (300 bps)

Operates in AUTODIN modes II, IV and V.
Uses ITA #2 code in JANAP-128 or ACP-127 formats.

(b) Type 2: <u>M</u>edium <u>S</u>peed <u>L</u>ine (MSL) (2400 bps)

Operates in AUTODIN mades II, IV and V.
Uses ASCII code and JANAP-128 or ACP-127 formats.

(c) Type 3: <u>H</u>igh <u>S</u>peed <u>L</u>ine (HSL) (9600 bps)

Operates in AUTODIN modes I and III.
Uses ASCII code and JANAP-128 format.

Therefore: 2 codes, ITA #2 and ASCII, are used; 2 formats, JANAP-128 and ACP-127, are used.

(9) Language Media Format (LMF)

Three types of LMF's are assumed:

(1) card format

(2) paper tape format

(3) magnetic tape format

Any LMF may be used with any message.

Assume an even distribution of LMF type for all line type messages.

(10) Channel Coordination

Type 1 Line

<u>Data receive</u> - recognize a steady mark (stop) signal as the idle pattern and a transition from mark to space as the start of the character frame.

<u>Data transmit</u> - two methods used;

(1) transmit data rhythmically with no external character step.

(2) transmit each character upon the receipt of a character step (release) pulse and maintain a steady mark condition upon completion of a character until the receipt of the next step pulse.

Assume that 1/2 of the Type 1 lines transmit with method 2.

Type 2 Line

Same as Type 1 line as far as coordination procedures are concerned.

Assume that 1/2 of the Type 2 lines transmit with method 2.

Type 3 Line

Data receive - data control characters used on a 84 character block basis. Receives idle pattern for synchronization, verification and channel checking purposes.

Data transmit - data control characters transmitted with each 84 character block. During idle periods, transmits the idle pattern.

(11) Average Holding Time ($\overline{HT}$) (by line speed)

HSL	$\overline{HT}$	=	20 seconds/msg.
MSL	$\overline{HT}$	=	65 seconds/msg.
LSL	$\overline{HT}$	=	500 seconds/msg.

(12) Message Distribution (by line speed)

Originated Line Speed	Messages Orig.	Terminated Line Speed HSL	MSL	LSL	TOTALS
HSL	360	144	1224	72	1800
MSL	3060	1224	10404	612	15300
LSL	180	72	612	36	900
TOTALS	3600	1440	12240	720	18000

(13) Distribution of Terminals (by line speed)

Line Speed	Messages	Erlangs	Terminals	(GOS)
HSL	1440	8	18	.001
MSL	12240	221	245	.01
LSL	720	100	117	.01
TOTALS	18000	329	380	N/A

(14) Message Accountability Distribution

Minimum Acc.	20%
Intermediate Acc.	20%
Historical/Recovery Acc. (H/R)	20%
Historical-Recovery-Traffic Acc. (H/R/T)	20%
Maximum Acc.	20%

(15) Accountability Functions

(a) Minimum Accountability

Buffer storage for active processing

Maintains delivery queues

Maintains channel queues

Releases messages (removes from intransit storage) upon receipt of acknowledge by receiving equipment.

Traffic statistics are kept, e.g.,

message counts

character counts

statistics on acknowledgements (ACK's, NACK's)

(b) Intermediate Accountability (Historical)

provides, in addition to minimum accountability facilities; historical records of predetermined processing events written to an on-line storage device. A journal of message events is kept.

(c) Historical/Recovery Accountability (H/R)

in addition to the above, it provides recording of messages on an Intransit Storage file. Some measure of status ledgering is performed to maintain the storage structure of the messages (or blocks).

(d) Historical/Recovery/Traffic Accountability (H/R/T)

in addition to the above, it provides recording of message blocks as received on separate recording medium. This allows retrieval and retransmission functions to be performed.

(e) Maximum Accountability

All functions of above accountability procedures plus added redundancy and storage capabilities. Reference files are kept. Complete status ledgers redundantly stored.

(16) Headers

An average of three 84 character blocks per message required for headers.

(17) Routing Indicators

Data Messages - an average of 2 RI's per message.

Narrative Messages - an average of 7 RI's per message.

Average of 5 characters per RI (data messages)

Average of 20 characters per RI (narrative messages)

(18) Routing Validation

Data Messages - includes checking the originating station identifier and serial number, checking each RI against the RI Table and formatting the outgoing RI on all messages to other M/S's.

Narrative Messages - includes all of the processes for data messages but requires examining the textual content of each message to find the addresses.

(19) Retransmission of 5% of all 84 character blocks is assumed.

Calculations

(1) Code Conversions

ASCII to ITA #2	=	864 messages/BH
ITA #2 to ASCII	=	864 messages/BH

TOTAL messages requiring code conversion	=	1,728 messages/BH
TOTAL Characters	=	32,348,150 char/BH
	=	8,986 char/sec

(2) Format Conversions

JANAP-128 to ACP-127	=	8100 messages/BH
ACP-127 to JANAP-128	=	8100 messages/BH
TOTAL messages requiring format conversion	=	16,200 messages/BH
TOTAL Characters	=	303,264,000 char/BH
		84,240 char/sec

(3) Language Media Conversions (LMF)

paper tape to card	=	2000
paper tape to magnetic tape	=	2000
card to paper tape	=	2000
card to magnetic tape	=	2000
magnetic tape to paper tape	=	2000
magnetic tape to card	=	2000
TOTAL messages requiring LMF conversion	=	12,000/BH
TOTAL characters	=	224,640,000/BH
	=	52,400/sec

(4) Header Validations and Generations

Total incoming messages = 3600/BH

Total headers = 3600/BH

Total blocks = 10800/BH

Total header characters requiring validation = 907,200/BH

= 252/sec

Total outgoing messages = 14400/BH

Total headers = 14400/BH

Total blocks = 43200/BH

Total header characters assembled for output = 3,628,800/BH

= 1,008/sec

(5) RI Validations and Generations

Data Messages

Total incoming messages = 2160/BH

Total RI's per message = 2

Rotal RI's = 4320/BH

Total characters = 21600/BH

= 6/sec

Total outgoing messages requiring RI's = 864/BH

Total RI's assembled = 1728/BH

Total characters (5 char/RI) = 8640/BH

= 2.4/sec

Narrative Messages

Total incoming narrative messages = 1440/BH

Total RI's per message = 7

Total RI's = 10080

Total characters examined to determine RI (plain language and textual addressing used)

= 18,144,000/BH

= 5,040/sec

Total outgoing messages requiring RI's = 576/BH

Total RI's assembled	=	4032/BH
Total characters (20 char/RI)	=	80,640/BH
	=	22.4/sec

(6) Traffic Statistics

Incoming messages	=	3,600/BH
Incoming characters	=	67,392,000/BH
Incoming blocks (84 characters)	=	802,296/BH
Retransmission (blocks)	=	40,114/BH
Outgoing messages	=	14,400/BH
Outgoing characters	=	269,568,000/BH
Outgoing blocks (84 characters)	=	3,209,143/BH
Retransmission (blocks)	=	106,457/BH
TOTAL incoming message blocks including retransmitted blocks	=	842,410/BH
TOTAL incoming characters	=	70,762,440/BH
	=	19,656/sec
TOTAL outgoing message blocks including retransmitted blocks	=	3,315,600/BH
TOTAL outgoing characters	=	278,510,400/BH
	=	77,364/sec

2.2 Breakdown Of M/S Functions

2.2.1 CIF Breakdown

2.2.1.1 Packet Switch

Incoming Traffic

Lines monitored	=	40
Character framing - includes conversion of data stream into 8-bit characters	=	37,800,000
Parity checks (one per character)	=	37,800,000
Packet framing - includes stripping of sync and control characters (10 characters/packet) and adding internal message identifiers (line ID, sequence number, packet number, etc. (10 characters)	=	75,600
Automatic packet retransmission request messages sent. - includes formatting a 80 character packet to be sent to transmitting station	=	3,600
SOM messages sent to MPF - includes time of arrival code, packet identifier code and precedence code if any	=	72,000
EOM messages sent to MPF - includes time of arrival code and packet identifier code	=	72,000
Packets transferred to HAF	=	72,000

Outgoing Traffic

Packets received from HAF	=	72,000
Packets framed for transmission - includes stripping internal identifier codes (retaining code for record keeping) and adding sync and control characters	=	72,000

Characters transmitted	=	37,800,000
Character parity check (or generation) on transmitted characters	=	37,800,000
Packets retransmitted	=	3,600
Transmit records to MPF - includes packet identifier code	=	75,600

2.2.1.2 Store and Forward M/S

Incoming Traffic

Lines monitored	=	170
Character framing - includes conversion of data stream into 5-8 bit characters depending on code used	=	45,360,000
Parity checks (one per character)	=	45,360,000
Block framing - includes stripping of sync and control characters (4 characters/84 character block) and adding internal messages identifiers (line ID, sequence number, block number, etc. 20 characters)	=	540,000
Parity checks (one per block)	=	540,000
Automatic block retransmission requests sent - includes formatting an 84 character block to be sent to the transmitting station	=	25,714
SOM messages sent to MPF - includes time of arrival code, block identifier code and precedence code	=	18,000
EOM messages sent to MPF - includes time of arrival code and block identifier code	=	18,000

Blocks transferred to CCF - includes those blocks which require character bit padding, code and format conversions	=	51,429
Blocks transferred to HAF directly - includes those blocks whose code and format is already compatible with the internal processing	=	462,857

<u>Outgoing Traffic</u>

Blocks received from CCF - includes only those message blocks that require code and/or format conversion	=	914,286
Blocks received from HAF - includes those message blocks for which no conversion is required	=	114,286
Blocks framed for transmission - includes stripping internal identifier codes (retaining code for record keeping)and adding sync and control characters	=	1,080,000
Characters transmitted	=	90,720,000
Character parity check (or generation) on transmitted characters	=	90,720,000
Blocks retransmitted	=	51,428
Transmit records sent to MPF - includes block identifier codes	=	1,080,000

2.2.1.3 Base Distribution M/S

<u>Incoming Traffic</u>

Lines monitored	=	380
Character framing (same conditions as S/F)	=	70,762,440
Parity checks (one per character)	=	70,762,440
Block framing (same conditions as S/F)	=	842,410

Parity check (one per block)	=	842,410
Automatic block retransmission requests (same conditions as S/F)	=	40,114
SOM messages sent to MPF (same conditions as S/F)	=	3,600
EOM messages sent to MPF (same conditions as S/F)	=	3,600
Blocks transferred to CCF (same conditions as S/F)	=	534,864
Blocks transferred to HAF (same conditions as S/F)	=	267,432

<u>Outgoing Traffic</u>

Blocks received from CCF (same conditions as S/F)	=	2,852,571
Blocks received from HAF (same conditions as S/F)	=	356,572
Blocks framed for transmission (same conditions as S/F)	=	3,209,143
Characters transmitted	=	278,510,400
Character parity check (same conditions as S/F)	=	278,510,400
Blocks retransmitted	=	106,457
Transmit records sent to MPF (same conditions as S/F)	=	3,209,143

2.2.2 <u>CCF Breakdown</u>

2.2.2.1 Packet Switch

No Compatibility Conversion Function operations required for the P/S system.

2.2.2.2 Store and Forward M/S

<u>Incoming Traffic</u>

Blocks received from CIF	=	48,857

Blocks requiring bit padding only (pad up to 8-bits)	=	0
Characters padded	=	0
Blocks requiring code conversion (conversion to ASCII)	=	48,857
Characters converted	=	4,104,000

Outgoing Traffic

Blocks received from MPF	=	290,571
Blocks requiring bit stripping - includes stripping bits from the internal 8-bit characters down to 7, 6, 5, etc., bit characters for transmission	=	0
Characters converted	=	0
Blocks requiring code conversion	=	97,714
Characters converted	=	8,208,000
Blocks requiring LMF conversion - includes stripping certain specialized characters, refram-ing the block lengths, and adding other specialized characters per block	=	685,714

2.2.2.3 Base Distribution M/S

Incoming Traffic

Blocks received from CIF	=	2,190,240
Blocks requiring bit padding only (pad up to 8-bits)	=	0
Characters padded	=	0
Blocks requiring code conversion (conversion to ASCII)	=	385,097
Characters converted	=	32,348,160

Outgoing Traffic

Blocks received from MPF	=	2,139,428
Blocks requiring bit stripping (same conditions as S/F)	=	0
Characters converted	=	0
Blocks requiring code conversion	=	385,097
Characters converted	=	32,348,160
Blocks requiring LMF conversion (same conditions as S/F)	=	2,139,428

2.2.3 HAF Breakdown

2.2.3.1 Packet Switch

Incoming Traffic

Header blocks received	=	72,000
Fields validated (12 per header)	=	864,000
Characters analyzed	=	3,600,000
Sequence number updates - includes checking for missing or out-of-sequence numbers	=	72,000
Service messages generated - assumes 5% of headers require service message to sending station	=	3,600
Service messages to OMF - assumes 10% of headers require special message to OMF	=	7,200

Outgoing Traffic

Header blocks received	=	72,000
Headers generated	=	72,000
Fields validated (12 per header)	=	864,000
Characters analyzed	=	3,600,000

Sequence numbers checked	=	72,000
Service messages generated - assumes 5% of headers require service messages to OMF	=	3,600

2.2.3.2 Store and Forward M/S

Incoming Traffic

Header blocks received	=	27,000
Fields validated (12 per header)	=	324,000
Characters analyzed	=	2,268,000
Sequence number updates (same conditions as P/S)	=	18,000
Service messages generated (same conditions as P/S)	=	900
Service messages to OMF (same conditions as P/S)	=	1,800

Outgoing Traffic

Header blocks received	=	54,000
Header generated	=	54,000
Fields validated (12 per header)	=	648,000
Characters analyzed	=	4,536,000
Sequence numbers checked	=	36,000
Service messages generated (same conditions as P/S)	=	1,800

2.2.3.3 Base Distribution M/S

Incoming Traffic

Header blocks received	=	3,600
Fields validated (12 per header)	=	43,200
Characters analyzed	=	302,400

Sequence number updates (same conditions as P/S)	=	3,600
Service messages generated - assumes 10% of local originated message headers requires service message	=	324
Service messages to OMF (same conditions as P/S)	=	360

Outgoing Traffic

Header blocks received	=	18,000
Headers generated	=	18,000
Fields validated (12 per header)	=	216,000
Characters analyzed	=	1,512,000
Sequence number checked	=	18,000
Service messages generated (same conditions as P/S)	=	1,800

2.2.4 MPF Breakdown

2.2.4.1 Packet Switch

Incoming Traffic

Header blocks received	=	72,000
RI's received	=	72,000
Characters analyzed (6 per RI) - includes Table look-up operations	=	432,000
Formatted delivery instructions	=	72,000
Accountability		
Entry messages - includes the following: redundant intransit storage, complete event journaling, reference files, status ledgering, and long term message storage	=	10,800

Exit messages - includes the following: active intransit storage, event journal files, and status ledgering	=	10,800
Transit messages - includes the following: active intransit storage with release of storage area upon acknowledgement of receipt by receiving station. Maintenance of delivery queues and channel queues on non-volatile storage.	=	52,020
Packets stored in non-volatile intransit storage	=	10,800
Characters stored (500 char/packet)	=	5,400,000
Journal events stored (10/packet) (non-volatile)	=	108,000
Packets stored in reference file (non-volatile)	=	10,800
Packets stored in active intransit storage	=	72,000
Characters stored in active storage	=	36,000,000
Acknowledgements received		
Maximum Acc. (4/packet)	=	43,200
Minimum Acc. (2/packet)	=	122,200
TOTAL	=	165,400

Outgoing Traffic

RI's formatted for delivery	=	61,200
Characters validated (6/RI)	=	367,200
Acknowledgements generated		
Entry Traffic (2/packet)	=	21,600
Exit Traffic (High-to-Low)	=	9,180
Transit Traffic (2/packet)	=	104,040
TOTAL	=	134,820
Packets removed from storage	=	61,200
Characters removed from storage	=	30,600,000

System Executive Functions

- include the following:

Message statistics updates

Input buffer checks

Output process checks

Monitoring line activity

Maintenance of line status tables

Directs storage processes-passes packets to be stored to OMF along with directions as to the type of storage media to be used.

2.2.4.2 Store and Forward M/S

Incoming Traffic

Header blocks received	=	27,000
RI's received	=	27,000
Characters analyzed (6/RI) - includes table look-up operations	=	162,000
Formatted delivery instructions	=	54,000
Accountability		
(a) Minimum (number of messages) - includes active intransit storage and maintenance of delivery queues and channel queues	=	3,600
(b) Intermediate (number of messages) - includes active intransit storage, maintenance of delivery and channel queues and creating event journals	=	3,600
(c) Historical/Recovery (number of messages) - includes all of above plus recording of messages on reference file and some status ledgering	=	3,600

(d) Historical/Recovery/Traffic (number of messages) - includes all of above plus recording of message blocks as received (non-volatile) complete journaling and ledgering (also stored on non-volatile media)	=	3,600
(e) Maximum (number of messages) - includes all of the above plus complete redundancy of all storage and long term storage (up to 30 days)	=	3,600
Blocks stored in non-volatile intransit storage	=	308,512
Character stored (100 char/block)	=	30,851,200
Journal events stored (15/message)	=	216,000
Messages stored in reference file	=	10,800
blocks	=	308,512
characters	=	30,851,200
Blocks stored in active intransit storage	=	514,286
Character in active intransit storage	=	51,428,600
Acknowledgements received		
Minimum (2/MSG)	=	7,200
Intermediate (2/MSG)	=	7,200
H/R (3/MSG)	=	10,800
H/R/T/ (4/MSG)	=	14,400
Maximum (4/MSG)	=	14,400
TOTAL ACK's	=	54,000

Outgoing Traffic

RI's formatted for delivery	=	37,800
Characters validated	=	226,800

Acknowledgements generated		
Minimum (2/MSG)	=	7,200
Intermediate (2/MSG)	=	7,200
H/R (2/MSG)	=	10,800
H/R/T (4/MSG)	=	14,400
Maximum (4/MSG)	=	14,400
TOTAL	=	54,000
Messages removed from intransit storage	=	7,200
Blocks	=	205,714
Characters	=	20,571,400

<u>System Executive Functions</u>

- include the same operations as in the Packet Switch.

2.2.4.3 Base Distribution M/S

<u>Incoming Traffic</u>

Header blocks received (data MSG)	=	2,160
Narrative messages received (RI validation for narrative messages requires examination of the entire message, character by character, to detect plain language and textual addresses)	=	1,440
RI's received (2 per data MSG) (7 per narrative MSG)	=	14,400
Character analyzed - includes 5 characters per RI (data MSG) 1800 characters per narrative MSG	=	2,613,600
Formatted delivery instructions	=	14,400
Accountability - includes same operations per category as in S/F M/S		
(a) Minimum (number of messages)	=	720

(b) Intermediate (number of messages)	=	720
(c) H/R (number of messages)	=	720
(d) H/R/T (number of messages)	=	720
(e) Maximum (number of messages)	=	720
Blocks stored in non-volatile intransit storage	=	481,372
Characters stored (100 char/block)	=	48,137,200
Journal events stored (15/MSG)	=	43,200
Messages stored in reference file	=	2,160
blocks	=	481,372
characters	=	48,137,200
Blocks stored in active intransit storage	=	802,286
Characters in active intransit storage	=	80,228,600
Acknowledgements received (same conditions as S/F)	=	10,800

Outgoing Traffic

RI's formatted for delivery	=	11,808
Characters validated (5/data MSG) (20/narrative MSG)	=	129,888
Acknowledgements generated (same conditions as S/F)	=	10,800
Messages removed from intransit storage	=	1,440
blocks	=	320,914
characters	=	32,091,400

System Executive Functions

- includes the same operations as in the Packet Switch

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL 1776-1976

☆U.S. GOVERNMENT PRINTING OFFICE: 1977-714-025/107